National Defence / Défense nationale

IMPLEMENTATION OF FFT AND PULSE COMPRESSION ROUTINES ON THE SPT FREQUENCY DOMAIN ARRAY PROCESSOR

by

V. Behroozi and A. Damini
Airborne Radar Section
Radar Division

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
REPORT NO.1041

PCN
021LA

September 1990
Ottawa

91-01784/

ABSTRACT

The Frequency Domain Array Processor (FDAP) is a VME compatible circuit board built by Signal Processing Technologies (SPT). The FDAP can process integer data arrays containing up to 8192 (32 bit) complex words or 16384 (16 bit) real words. It is capable of 400 Million Operations Per Second (MOPS) with a maximum Input/Output (I/O) rate of four billion bits per second. It also has a double buffered memory architecture permitting I/O transfers to occur in parallel with data processing. The FDAP can be hosted by an IBM PC/AT-compatible computer using a bus adaptor interface available from BIT3 Computer Corp.

The FDAP board is based upon SPT's DASP/PAC chip set. This chip set and the various system architectures which can be built around it are reviewed. The FDAP board and its associated development system are also reviewed. The ease of implementation of typical radar signal processing functions on the FDAP board are then examined. Fast Fourier Transform and pulse compression routines are implemented via a supplied user interface as well as a high level language (C). The results are examined and comments on the FDAP and its associated system development tools are made.

RÉSUMÉ

Le Processeur Parallèle dans le Domaine des Fréquences, (PPDF) est une carte électronique compatible avec VME fabriquée par Signal Processing Technologies (SPT). Le PPDF peut traiter des vecteurs de données entières contenant jusqu'à 8192 (32 bits) codes complexes ou 16384 (16 bits) codes réels. Il est capable de réaliser 400 Millions Operations Par Seconde (MOPS) avec un taux maximum d'Entrée/Sortie (E/S) de quatre milliards de bits par seconde. Il possède aussi deux mémoires tampons permettant le traitement et la transmission de données simultanément. Le PPDF fonctionne sur un ordinateur compatible avec un IBM PC/AT à l'aide d'une interface adaptive bus disponible chez la corporation BIT3 Computer.

La carte PPDF est basée sur la famille de puces DASP/PAC de SPT. Cette famille et les systèmes d'architecture variés qui peuvent y être associés sont revisés. La facilité d'implantation de fonctions typiques d'analyse de signaux radars sur la carte PPDF est ensuite examinée. Les transformées de Fourier Rapides et routines de compression d'impulsions sont implantés par l'usager via une interface et un haut niveau de langage (C). Les résultats sont examinés et la carte PPDF, ainsi que les outils de développement de système qui lui sont associés, sont commentés.

EXECUTIVE SUMMARY

State of the art Synthetic Aperture Radars (SAR) transmit high bandwidth pulses and subsequently require extremely high sampling rates. The processing rates required to accommodate the resulting data rates are also very high. The advances in board level products for high-speed standard busses has reached the point where it may be possible to replace current processors with ones occupying much smaller volumes, which are more suitable for an airborne environment. This report examines the use of one such card for application to a subset of operations required in such processors.

The board-level processor examined in this study is the Frequency Domain Array Processor (FDAP), a VME compatible circuit board built by Signal Processing Technologies (SPT). The FDAP can process integer data arrays containing up to 8192 (32 bit) complex words or 16384 (16 bit) real words. It is capable of 400 Million Operations Per Second (MOPS) with a maximum I/O rate of four billion bits per second. The FDAP board is based upon SPT's DASP/PAC chip set. This chip set and the various system architectures which can be built around it are reviewed. The FDAP board and its associated development system are also reviewed. The ease of implementation of typical radar signal processing functions on the FDAP board are then examined. Fast Fourier Transform and pulse compression routines are implemented via a supplied user interface as well as a high level language (C). The results are examined and comments on the FDAP and its associated system development tools are made.

TABLE OF CONTENTS

		Page
	ABSTRACT	iii
	RÉSUMÉ	iii
	EXECUTIVE SUMMARY	v
	TABLE OF CONTENTS	vii
	LIST OF FIGURES	ix
	LIST OF TABLES	ix
1.0	INTRODUCTION	1
1.1	The DASP/PAC Chip Set	2
1.1.1	The Digital Array Signal Processor (DASP)	2
1.1.2	The Programmable Array Controller (PAC)	3
2.0	SYSTEM ARCHITECTURES	6
2.1	Recursive Dual Memory System-Core	6
2.2	Recursive Dual Memory System-I/O Buffered	7
2.3	Recursive Single Memory System	8
2.4	Cascaded System	8
2.5	A Digital Filtering System	9
3.0	SYSTEM DEVELOPMENT TOOLS	11
3.1	Software Simulators	11
3.2	Hardware Evaluation Module (EVM)	12
3.3	VME Interface and Personal Computer	13
4.0	FREQUENCY DOMAIN ARRAY PROCESSOR (FDAP)	14
4.1	Architecture	14
4.2	System Components	14
5.0	PROGRAMMING THE FDAP	17
5.1	Memory Map	17
5.2	Control and Status Registers	18
5.2.1	Memory Map	18
5.3	PAC Registers	19
5.3.1	PAC Instructions	19
5.3.2	PAC Control Registers	20
5.4	Instruction Format	22
5.4.1	Node Type Field	22
5.4.2	Function Set	25

Page

6.0 FDAP PROGRAMMING USING A USER INTERFACE 26
6.1 Radix-4 FFT Program 26
6.2 Pulse Compression Program Using Radix-4 FFTs 33
6.2.1 Theory of Pulse Compression 33
6.2.2 Pulse Compression on the FDAP board 34
6.2.3 Pulse Compression Program 34
7.0 FDAP PROGRAMMING USING A HIGH LEVEL LANGUAGE .. 39
8.0 CONCLUSIONS 39
9.0 APPENDIX 1 - FDAP C-LANGUAGE PROGRAMS 40
9.1 1024-Point Radix-4 Complex FFT Program 40
9.2 1024-Point Radix-4 Pulse Compression Program 44
10.0 REFERENCES 49

LIST OF FIGURES

Page

Figure 1.1: DASP Block Diagram and I/O Values 2
Figure 1.2: DASP System 3
Figure 1.3: PAC Block Diagram 4
Figure 1.4: Dual Memory Recursive System With I/O Buffers 5
Figure 1.5: Recursive System With I/O Buffers-Memory Reallocation 5
Figure 2.1: Core Dual Memory Recursive Stage 6
Figure 2.2: Core Stage-Reverse Direction 7
Figure 2.3: Single Memory Recursive System 8
Figure 2.4: Cascaded System 9
Figure 2.5: Digital Filtering in the Frequency Domain 9
Figure 2.6: Overlap/Discard Fast Convolution 10
Figure 2.7: Overlap/Discard Recursive System With I/O Buffers 10
Figure 3.1: DASP System Development Cycle 11
Figure 3.2: EVM Block Diagram 13
Figure 4.1: FDAP Block Diagram 16
Figure 5.1: Memory Map 18
Figure 5.2: Control and Status Register Memory Map 19
Figure 5.3: PAC Register Memory Map 20
Figure 5.4: PAC Instruction Format 22
Figure 5.5: a) Radix-4, 16-Point DIF FFT Flow Diagram 24
b) Radix-2, 16-Point DIF FFT Flow Diagram 24
Figure 6.1: Time Domain Data from Simulator (Memory B) 29
Figure 6.2: Twiddle Factors (Auxiliary Memory X1) 30
Figure 6.3: Harris-Blackman Window (Auxiliary Memory X2) 31
Figure 6.4: Frequency Domain Data (Memory C) 32
Figure 6.5: Pulse Compression Algorithm 33
Figure 6.6: Linear FM Pulse - Time Domain (Memory A) 36
Figure 6.7: Matched Filter Impulse Response - Frequency Domain (Auxiliary Memory X2) 37
Figure 6.8: Power Spectrum of Compressed Pulse (Linear Data from Memory D) 38

LIST OF TABLES

Table 5.1: PAC Node Type 23
Table 5.2: DASP Function Set 25

1.0 INTRODUCTION

Signal Processing Technologies (SPT) has recently introduced a new generation of Digital Signal Processing (DSP) VLSI high performance integrated circuits. These devices, part of the HDSP66 device family, incorporate innovative architectures and a high-performance, 2 micron CMOS process. They are superior to currently available devices in terms of speed, flexibility, power dissipation, level of integration, and cost [1]. The capability of the devices to carry out 400 Million Operation Per Second (MOPS) makes high performance solutions for Fast Fourier Transform (FFT) intensive applications such as those found in radar possible.

The product categories in the HDSP66 family include [5]:

i) Processor chips,
ii) Memory chips and modules,
iii) Simulation and software development tools, and
iv) Processor boards.

These categories are designed to complement each other in any development environment.

This report summarizes work performed on the development of a digital Fourier Transform processor for FFT and pulse compression applications. The report first reviews the DASP/PAC chip set and the various system architectures which can be configured around it. The system development tools supplied by SPT and the FDAP, a high-speed signal processing board built by SPT and based on the DASP/PAC chip set, are then examined. Finally, some practical implementations of FFT and pulse compression routines are presented.

1.1 The DASP/PAC Chip Set

The HDSP66 device family is based upon the Digital Array Signal Processor (DASP) and Programmable Array Controller (PAC) chip set. The DASP and the PAC allow FFT-based DSP systems to be implemented which process at data rates up to 80 MHz in real time. Discrete Fourier Transform, spectrum analysis, digital filters, correlations, convolutions, and adaptive filters based upon FFT techniques are possible [1]. DASP/PAC-based systems can compute a complex FFT of up to 64K points. Furthermore, multiple DASP/PAC chip sets can be combined to enhance performance to the desired level. For example, one DASP/PAC chip set can compute a 1024 point complex FFT in 131 microseconds while five chip sets can execute the same FFT in 26.2 microseconds [5].

The DASP/PAC chip set allows several FFT-based system architectures to be configured. These systems typically require only several PAC instructions to execute a DSP function such as an FFT.

1.1.1 The Digital Array Signal Processor (DASP)

The DASP, illustrated in Figure 1.1 [1], is a very high-speed, block floating-point array processor. It performs up to 400 MOPS such as multiplications and additions, when performing FFT specific and general-purpose operations on arrays of data. It is capable of a maximum I/O rate of four billion bits per second. The DASP performs functions consisting of multiple arithmetic operations. The operations are carried out on two sets of four 32-bit complex values or two sets of eight 16-bit real values every machine cycle. The lower limit on the machine cycle, which is the number of clock cycles needed to execute an instruction, is 100 nanoseconds (four clock cycles at forty MHz).

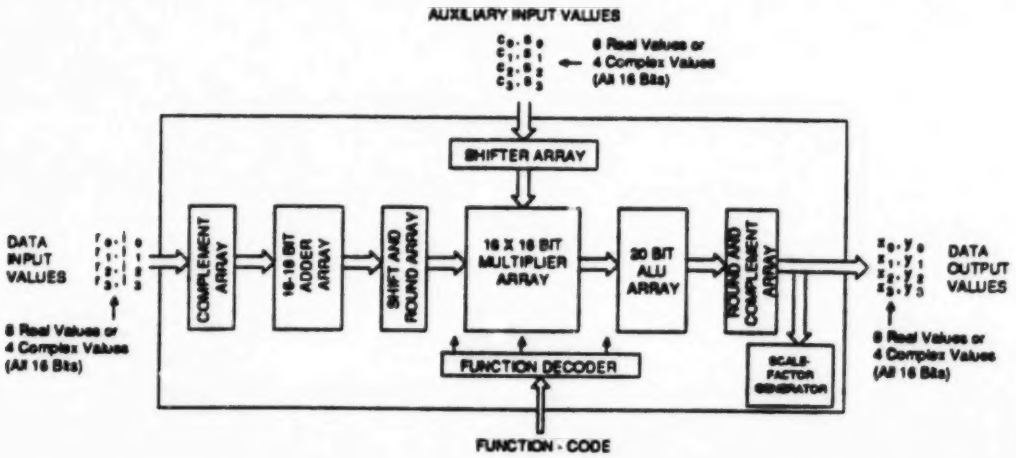

FIGURE 1.1: DASP BLOCK DIAGRAM AND I/O VALUES

The values input to the DASP and output from it are 16-bit fixed point numbers, represented in fractional two's complement arithmetic. To preserve an adequate signal-to-noise ratio as intermediate operations are carried out, intermediate values can grow to 20 bits. These same values are rounded to 16 bits upon transfer to the output busses. In fixed-point machines scaling is applied to prevent overflow. This must be taken into consideration as there is often an accompanying loss in signal-to-noise ratio.

1.1.2 The Programmable Array Controller (PAC)

The Programmable Array Controller (PAC) is used in conjunction with the DASP to configure FFT-based DSP systems. The DASP/PAC chip set can be combined with off-the-shelf single-port memories, as shown in Figure 1.2 [2], to develop DSP systems. The PAC acts as a local system manager. It contains a program memory which is initialized by a user-downloaded program. A PAC program, defined to implement either a FFT or a FFT-based function, is typically ten to twenty instructions long. Conventional DSP microprocessors, on the other hand, usually require on the order of ten to one hundred times as many instructions to carry out similar functions.

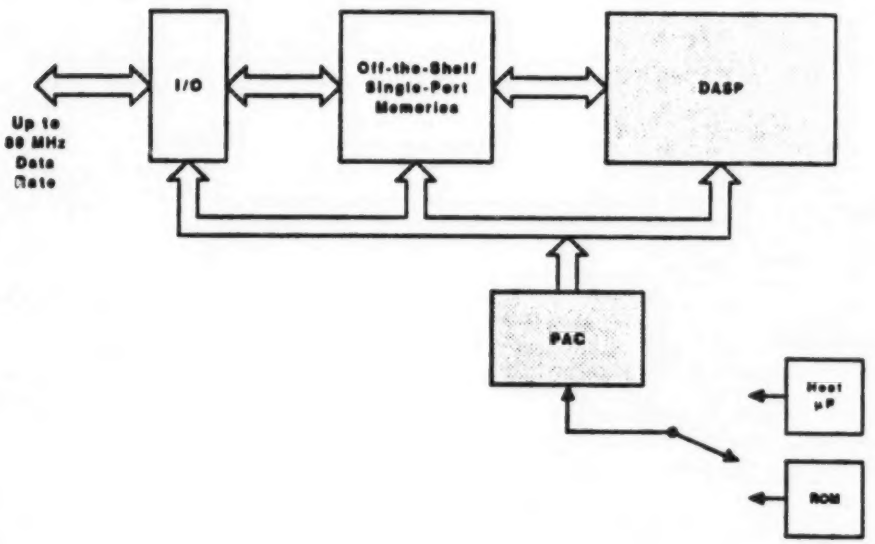

FIGURE 1.2: DASP SYSTEM

A block diagram of the PAC architecture is illustrated in Figure 1.3 [2]. The PAC contains 5 independent address generators each of which produces a 16-bit address every 25 nanoseconds. The generators, IAS, OAS, RAS, WAS and XAS, produce the address sequences for the input and output memories and the three DASP data memories. This is done in parallel, as shown in Figure 1.4 [1]. This allows the PAC to access data arrays up to 64K-points in size. The address sequences generated range from sequential for conventional array operations to FFT-specific for the various passes of the Decimation In Frequency (DIF) FFT algorithms. The FFT-specific address sequences are applied to the data and twiddle factor memories for radix-4, radix-2, and mixed radix-4/radix-2 FFT functions.

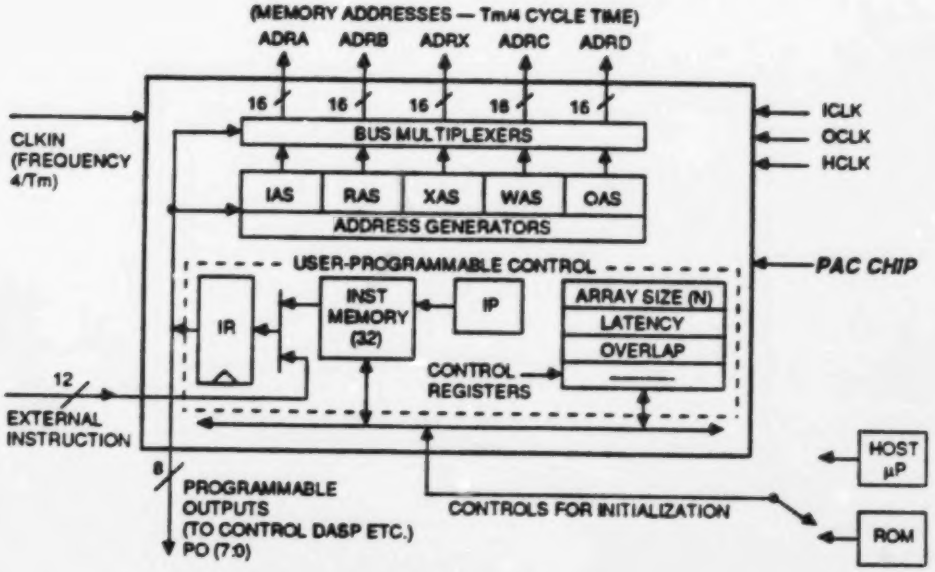

FIGURE 1.3: PAC BLOCK DIAGRAM

During the execution of a typical DASP function it is often necessary to reallocate the data memories. The PAC supports this requirement, as shown by Figures 1.4 and 1.5 [1], by integrating a bank of bus multiplexers. The various address sequences, produced by the five address generators, can thus be programmed to appear on any of the address busses ADRA, ADRB, ADRC, ADRD, and ADRX. Furthermore, each address generator has a memory write enable control signal associated with it which appears at the appropriate memory control output pins.

The PAC contains a 32 word, 20 bit user-loadable instruction RAM within which DASP instructions can be loaded. These instructions affect the address sequence required

of the address generators, and the programming of the address generators to the desired address buses. The PAC contains several user-loadable control registers which allow several system configurations to be defined. These control registers contain system configuration parameters which are discussed in Section 5.3.2. The PAC is typically initialized and hosted by an additional microprocessor. Alternatively, the PAC can autobootload itself from an external ROM.

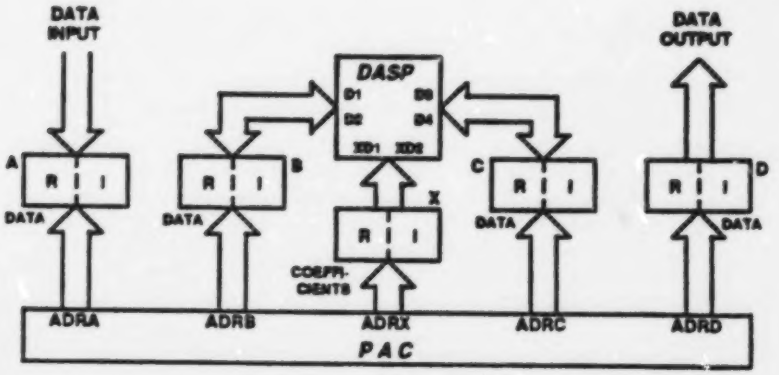

FIGURE 1.4: DUAL MEMORY RECURSIVE SYSTEM WITH I/O BUFFERS

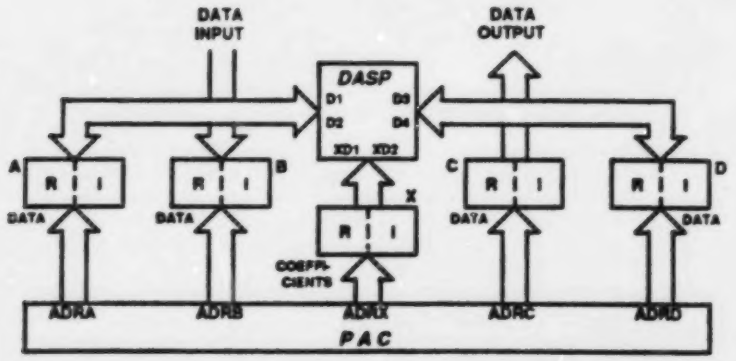

FIGURE 1.5: RECURSIVE SYSTEM WITH I/O BUFFERS-MEMORY REALLOCATION

2.0 SYSTEM ARCHITECTURES

The DASP/PAC chip set allows many FFT-based DSP problems to be solved using a variety of DSP algorithms and system configurations. The three basic system architectures supported by the chip set are recursive dual memory, recursive single memory, and cascaded [1]. The possible architectures are not restricted to the configurations herein presented but may be tailored to the application.

2.1 Recursive Dual Memory System-Core

The recursive architecture executes DSP algorithms on a single DASP/PAC stage by carrying out multiple passes of the data array through the DASP. The architecture of a core, recursive dual memory system is given in Figure 2.1 [1]. The PAC controls the system and thus the addressing of the three memories, B, C and X. Each of these memories can contain N words of complex data. N is the size of the array which is passed through the DASP.

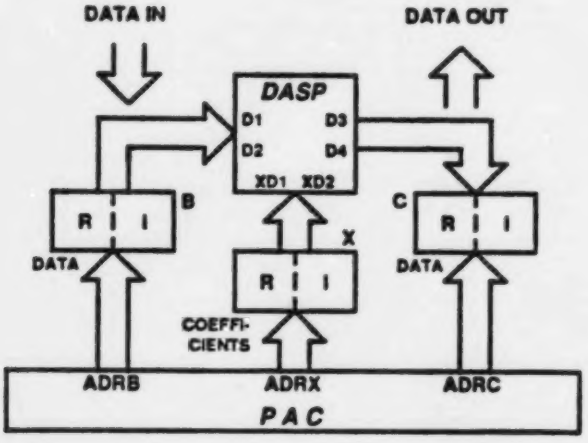

FIGURE 2.1: CORE DUAL MEMORY RECURSIVE STAGE

The initiation of any system function involves the input address being programmed to appear on the ADRB bus so that N data samples can be loaded in memory B. The DASP processing then commences with the PAC sending an instruction code to the DASP to implement the desired function. The first PAC instruction passes the data from memory B to memory C by providing the read address sequence on the ADRB bus and the write address sequence on the ADRC bus. An auxiliary address sequence is programmed on the ADRX bus to allow the DASP to read from memory X. The current function programmed

on the DASP is applied to all N points being passed. Upon completion of the data transfer, the PAC reverses the roles of memories B and C through its on-chip address multiplexer. This allows data to flow in the opposite direction as illustrated in Figure 2.2 [2].

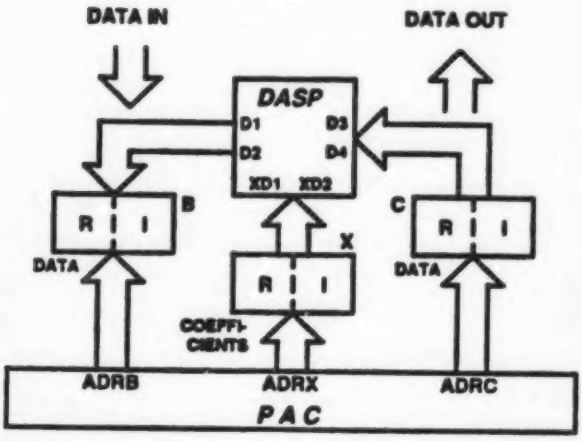

FIGURE 2.2: CORE STAGE-REVERSE DIRECTION

The data passes back and forth between memories B and C through the DASP. Each pass corresponds to one of the functions comprising the user's algorithm.

2.2 Recursive Dual Memory System-I/O Buffered

In the recursive dual memory system-core architecture, processing resources are idle while I/O is being performed. This makes the system undesirable for real-time applications. An architecture for a real-time DSP system would ideally take advantage of all five PAC address busses. The double-buffered configuration shown in Figure 1.4 makes use of such an architecture. Two additional memories are used for input and output while three other memories are used for the DASP data processing functions. In Figure 1.4 it can be seen that while the (t+1)th frame of data is being input in memory A, the (t-1)th processed data frame is being output from memory D. Memories B and C act as both read and write memories for recursive execution of the various functions comprising the DSP algorithm being executed on data frame t.

2.3 Recursive Single Memory System

To reduce the system cost, size, and power associated with the DASP/PAC-based systems we are considering, a system architecture configured around a single data memory can be utilized. Such a system is shown in Figure 2.3 [1].

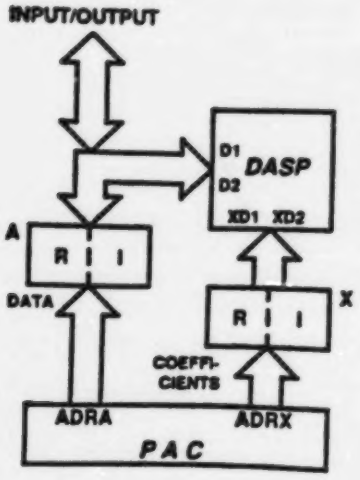

FIGURE 2.3: SINGLE MEMORY RECURSIVE SYSTEM

A single memory recursive system interleaves both the read and write operations, corresponding to the execution of the various passes of the in-place DIF FFT algorithm, on one data memory. The configuration makes use of only one of the DASP I/O ports and two of the PAC address busses. The DASP performance is instantaneously halved due to the I/O bottleneck on the DASP data port being used. The implementation of a function on this system can be explained by way of Figure 2.3. A data array is input in memory A by programming the input address sequence to the ADRA bus. Multiple DASP functions are then carried out by passing the data array through the DASP. The read and write addresses are interleaved on the ADRA bus during each of these passes. After the final pass the processed data is transferred out of memory A by programming the output address sequence on the ADRA bus.

2.4 Cascaded System

The effective throughput of a DASP/PAC-based system can be increased by cascading DASP/PAC stages. Such a system can process complex data at a 40 MHz rate

and real data at an 80 MHz rate. A cascaded system with multiple core recursive stages acting as the inner nodes and two I/O-buffered recursive stages acting as the end nodes is given in Figure 2.4 [1].

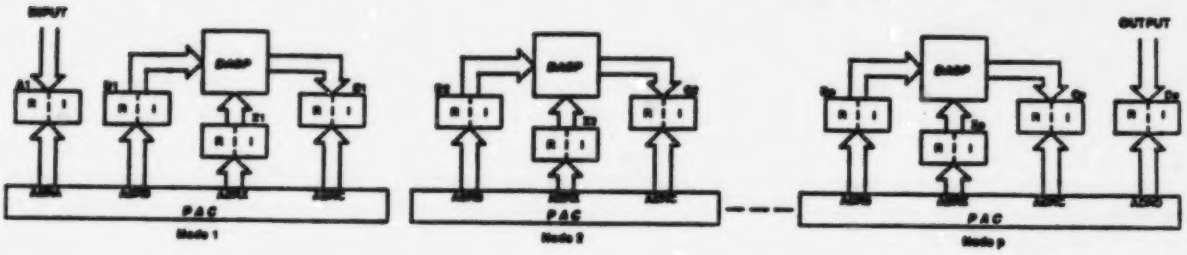

FIGURE 2.4: CASCADED SYSTEM

The algorithm is partitioned over these nodes. For example, the P passes of an FFT may be distributed over P processing nodes. Each node in a cascaded system executes the same function on successive data arrays. Each PAC contains two control registers which are used to initialize the type of nodes which are neighbour to it. This allows the PAC to generate the proper read and write address sequences for its left and right neighbouring nodes.

2.5 A Digital Filtering System

Digital filtering is a signal processing application which lends itself towards implementation on an I/O-buffered recursive system. Digital filtering involves the linear convolution of a long sequence with a filter of finite length. The linear convolution may be implemented by partitioning the long sequence into data arrays of length N. This partitioned sequence is then processed using the fast convolution algorithm shown in Figures 2.5 and 2.6 [1]. In the fast convolution algorithm the next data array must be overlapped with the previous data array by an amount at least equal to the filter length, K. Two overlap memories, OVA and OVB, are added to the system to hold the overlapped points as indicated in Figure 2.7 [1].

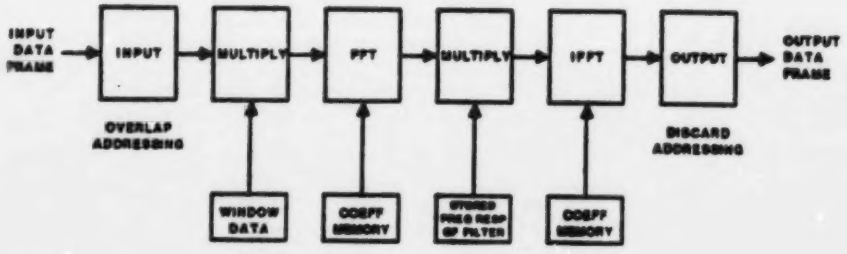

FIGURE 2.5: DIGITAL FILTERING IN THE FREQUENCY DOMAIN

As each set of N data points is output, the first K values in the data array must be discarded. This fast convolution technique is referred to as the overlap/discard method.

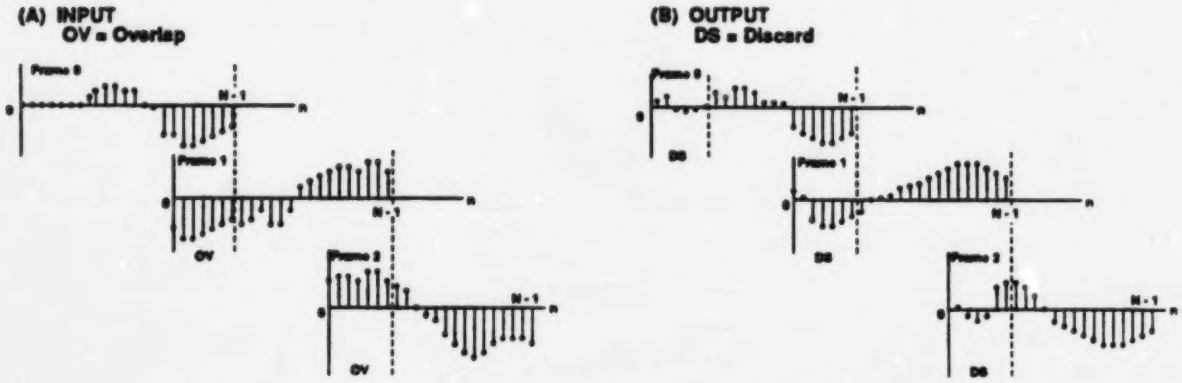

FIGURE 2.6: OVERLAP/DISCARD FAST CONVOLUTION

The PAC can be programmed for fast convolution through overlap/discard special addressing sequences, control signals, and a special control register referred to as K which is used to set the overlap size K. A recursive architecture configured as a digital filter system is shown in Figure 2.7.

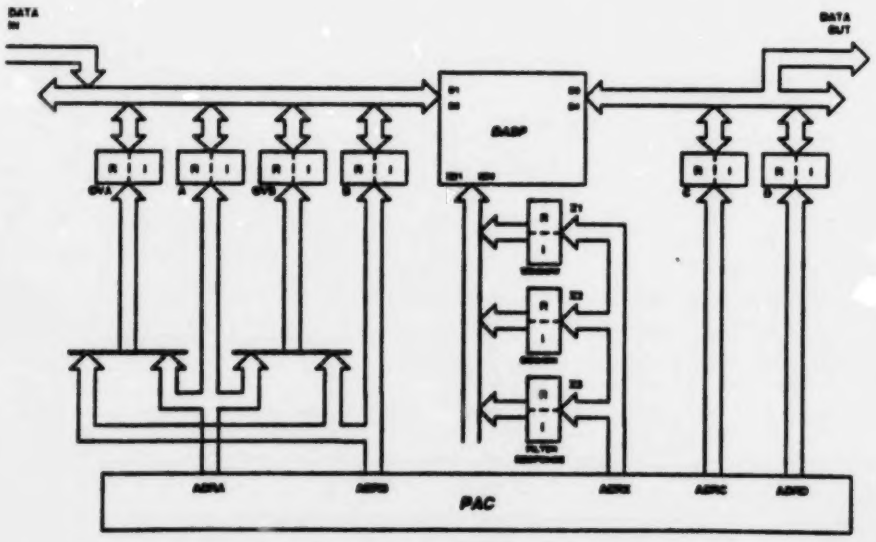

FIGURE 2.7: OVERLAP/DISCARD RECURSIVE SYSTEM WITH I/O BUFFERS

3.0 SYSTEM DEVELOPMENT TOOLS

The development of a DASP/PAC-based system is simplified by the use of system development tools provided by SPT. These tools include software simulators, user guides and a hardware evaluation module. The development cycle involved in designing and implementing a DASP/PAC-based system is illustrated in Figure 3.1 [1].

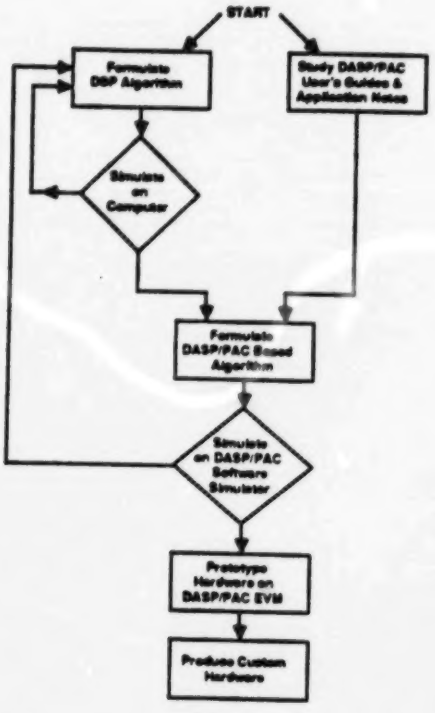

FIGURE 3.1: DASP SYSTEM DEVELOPMENT CYCLE

3.1 Software Simulators

SPT provides software simulators for the DASP/PAC chip set configured in a recursive dual memory-I/O buffered system. These simulators run on IBM PC and VAX/VMS systems. The key features of these simulators are listed below. The IBM PC-based simulator SIM.EXE, has a user interface compatible with that of the Hardware Evaluation Module (EVM) board, which is discussed in Section 3.2.

The VAX/VMS-based simulator is a stand-alone simulation tool. It can also be integrated into a larger simulation by being called as a subroutine. The VAX/VMS-based simulator was not used in this study.

The key features of the IBM PC based simulator include [2]:

i) A menu-driven user interface,
ii) A capability to generate data arrays consisting of two sinusoids,
iii) The DSP algorithm being defined through graphical models of the PAC instruction memory and control registers,
iv) Algorithm execution being carried out on a model of the DASP,
v) Graphical displays for data arrays, and
vi) I/O file capability for data arrays and programs.

The key features of the VAX/VMS based simulator include [2]:

i) Coding in "C" language to allow linking with user defined system simulations,
ii) Algorithm execution being carried out on an integrated model of the DASP/PAC chip set, and
iii) Corresponding floating point computations being available for examination of errors due to scaling and round-off.

3.2 Hardware Evaluation Module (EVM)

The EVM board is a DASP/PAC-based digital signal processing system implemented on a Eurocard SU-9 standard size board for VME bus systems. FFT-based signal processing algorithms can be implemented on the EVM to process data at rates currently available in VME bus products. The EVM allows the designer to prototype and evaluate a proposed DSP system. A block diagram of the EVM is given in Figure 3.2 [1]. A more detailed description of the EVM is found in Section 4.0.

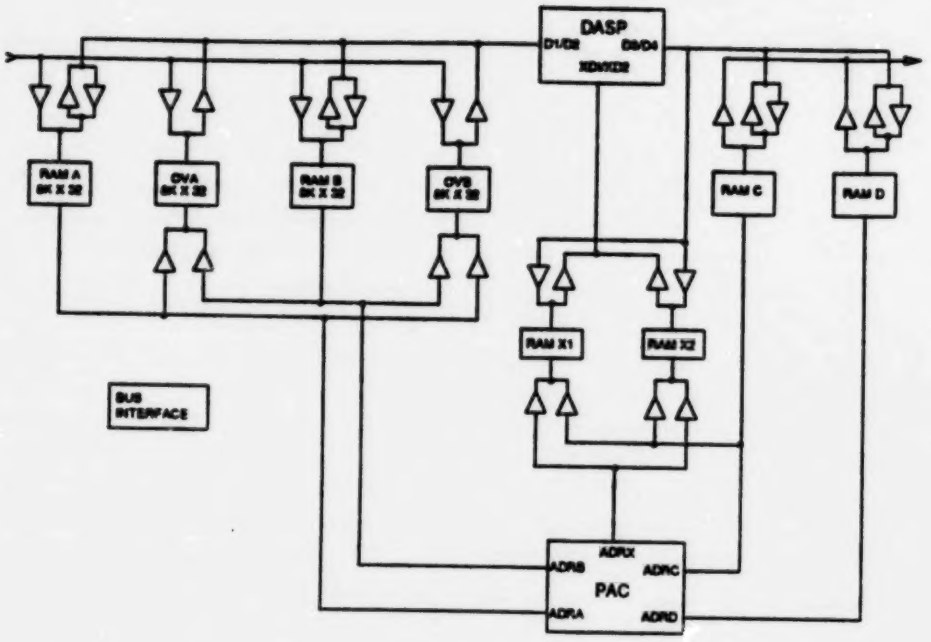

FIGURE 3.2: EVM BLOCK DIAGRAM

The EVM board may be controlled in one of two ways:

i) Using a supplied user interface, EVM.EXE, which is identical to that of the IBM PC based simulator, and
ii) Using a High Level Language.

3.3 VME Interface and Personal Computer

The VME interface allows parallel communication between the EVM and the IBM AT compatible personal computer. It consists of two cards and an interconnecting cable. One card resides with the evaluation system on the VME bus and the other is located in the AT computer. Address mapping permits the PC-AT to directly address VME memory on the BIT3 card in the VME chassis as though it were local memory. Communications between the two systems is via random access memory reads and writes. The PC-AT can execute code residing in memory on the VME bus. The jumpers on the PC-AT adaptor card establish various viewports or windows within the PC-AT memory and I/O space that map directly to the VME memory, adaptor dual port memory, and adaptor I/O.

Software on the AT allows control of the EVM from the computer keyboard. Data memories on the EVM may be observed and modified. Files stored on the computer may be transferred to the EVM and results may be stored in new files.

4.0 FREQUENCY DOMAIN ARRAY PROCESSOR (FDAP)

The Frequency Domain Array Processor (FDAP) is a DASP/PAC-based VME-compatible circuit board used for prototyping high speed DSP systems. The board contains 8 banks of 8K by 32 bit high speed static memory and control logic interconnected to enable FFT-based signal processors to be implemented.

The FDAP can process integer data arrays containing up to 8192 (32 bit) complex words or 16384 (16 bit) real words. The J1/P1 and J2/P2 connectors on the FDAP circuit board, which conform to the Eurocard SU-9 standard size, are standard VME interfaces. The J3/P3 connector is a custom-defined, high speed I/O interface through which multiple FDAPs may also be cascaded.

4.1 Architecture

The FDAP architecture is based upon the dual memory recursive or "ping-pong" architecture outlined in Sections 2.1 and 2.2. The FDAP also includes overlap/save memories, dual coefficient memories and additional data memories so that a double-buffered architecture can be enabled.

A block diagram of the FDAP is given in Figure 4.1 [5]. The main components comprising the architecture of the board are [5]:

i) The bus interface,
ii) Ping-Pong memories (A and D, B and C),
iii) Overlap memories (OVA and OVB),
iv) Coefficient memories (RAM X1 and RAM X2),
v) I/O interface, and
vi) DASP/PAC chip set.

4.2 System Components

The VME bus interface consists of the J1/J2 connectors. These connectors allow the FDAP to communicate with an external VME master. These communications include loading of the PAC control and instruction registers, and the reading and writing of data to and from the memory banks. The bus interface also allows the seven VME bus interrupts to be used to signal external interrupt handlers when the PAC reaches a predetermined point in the execution of a program.

The FDAP's ping-pong memory consists of four banks of 8Kx32 bit memory, A, B, C and D. It is referred to as ping-pong memory because data is passed recursively back and forth between pairs of memory banks. The PAC's ADRA, ADRB, ADRC, and ADRD address busses are used to address the four banks as shown in Figure 4.1. Banks A and B may be programmed to be either input memories or DASP read/write memories. If they are read/write memories their data busses are subsequently gated onto the DASP D1 and D2 data busses. Similarly, banks C and D may be programmed to be either output memories or DASP read/write memories.

When a fast convolution filtering algorithm using the overlap/discard technique is to be implemented, two banks of 8Kx32 bit memory are used to hold the overlapped portions of the input data. These two memory banks, OVA and OVB, support overlaps of up to 2K-points.

Coefficient memories X1 and X2, also referred to as auxiliary memories, are each 8Kx32 bit high-speed static memory banks. The DASP's auxiliary data bus may be routed to either auxiliary memory and the remaining memory may be concurrently written to from the DASP's D3/D4 bus. The two most significant bits of the PAC programmable output, which are discussed in Section 5.4, control the routing to the coefficient memories. Bit 6 determines the routing of the DASP auxiliary data bus while bit 7 enables the memory routed to the DASP D3/D4 bus to be written to.

The high-speed I/O interface is implemented through a 96 pin J3 connector on the FDAP. The interface consists of a 32 bit input bus, 32 bit output bus and control and status signals. Both busses are 32 bit complex. The input bus may be gated directly into either memory A or B or one of the overlap memories. The output bus may originate from either memory C or D.

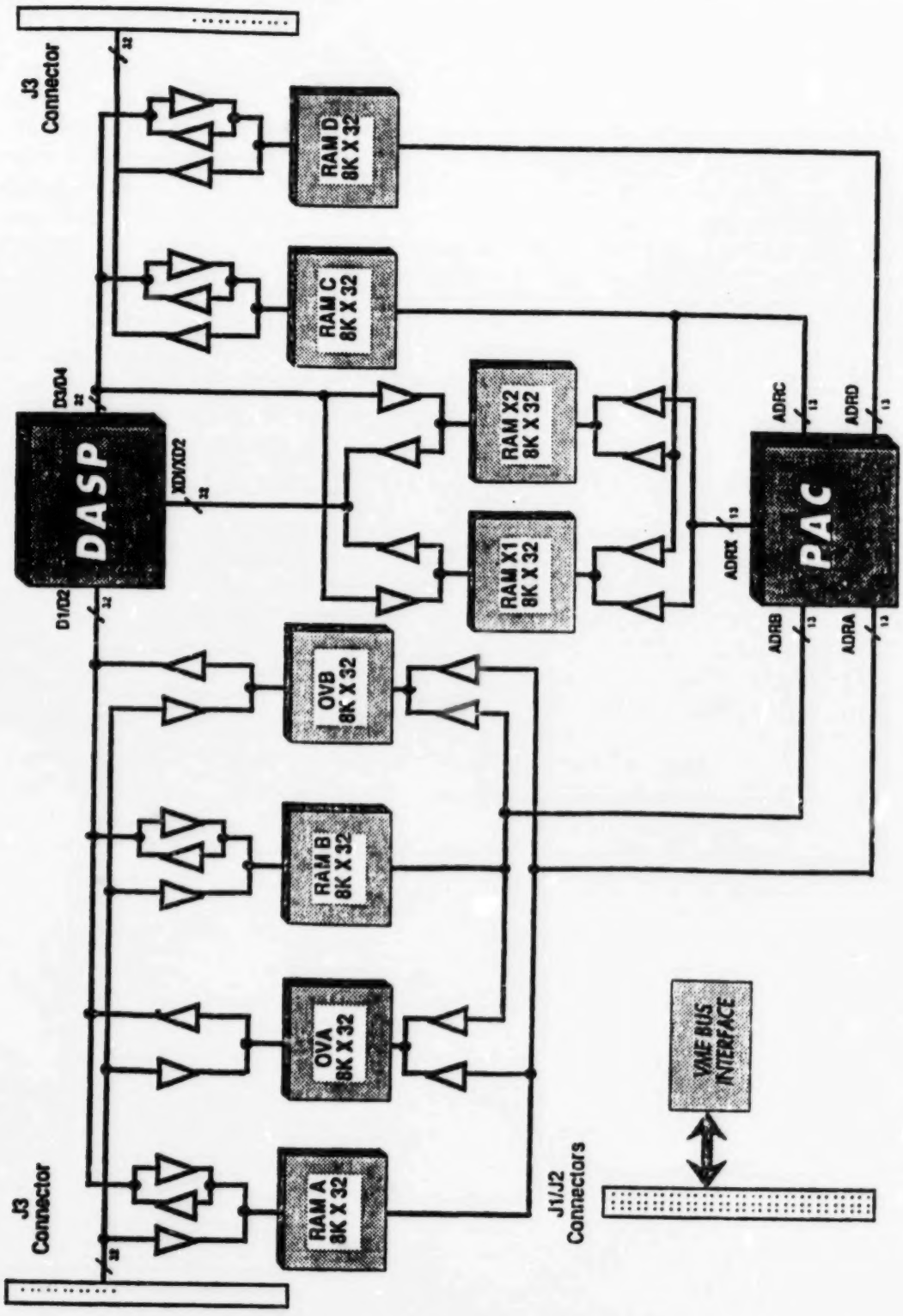

FIGURE 4.1: FDAP BLOCK DIAGRAM

5.0 PROGRAMMING THE FDAP

The FDAP memory banks and control and status registers are memory mapped to occupy 64K of VME memory space. The processing memories occupy the upper 32K bytes of FDAP address space. At any given moment, one of the eight 32K byte processing memories can be bank selected to occupy this space.

5.1 Memory Map

The VME memory map for the FDAP is illustrated in Figure 5.1 [5]. The control and status registers may be jumpered on the BIT3 card (which is located in the VME chassis) to occupy any one of the 16 different locations indicated. In our example Control/Status Location#1 is made use of and is expanded on in Figure 5.2 [5]. The FDAP base address must be set to a 64K byte boundary corresponding to an unused block in the PC-AT. The remote bus RAM jumper setting permits the PC-AT to read and write to the BIT3 memory card in the PC and thus the BIT3 VME adaptor card and processor card residing in the VME chassis. The HI and LO RAM jumpers (on the BIT3 card in the VME chassis) select the address range that the PC-AT will reference in its address space when it wants to read or write to memory in the VME bus. The remote RAM LO jumper sets the starting PC-AT address and the remote RAM HI selects the ending address. For this board the start address is D0000 hex and the end address is DEEEF hex. Data is transferred in 16 bit blocks to the FDAP as each FDAP memory location contains 16 bits.

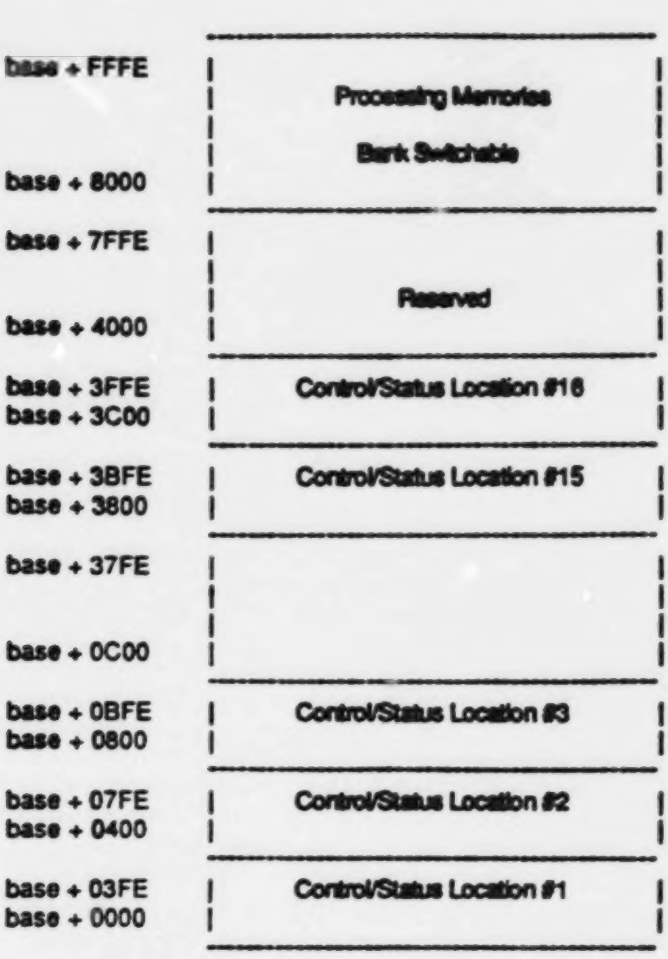

FIGURE 5.1: MEMORY MAP

5.2 Control and Status Registers

5.2.1 Memory Map

The memory map for the FDAP's control and status registers, Location#1 is illustrated in Figure 5.2. Most of the registers can be both read from and written to. The Cbase address for the Control/Status location is set via jumpers on the BIT3 card in the VME chassis. The PAC control registers are expanded on in Section 5.3 and Figure 5.3 [5]. The control and status registers occupy five words starting at memory location Cbase+200 Hex. These registers allow the user to set up interrupts, PAC execution modes, map in or out of various processing memory banks, and set up methods for handling scale factors for block floating point. For our board Cbase is set to D0000 hex.

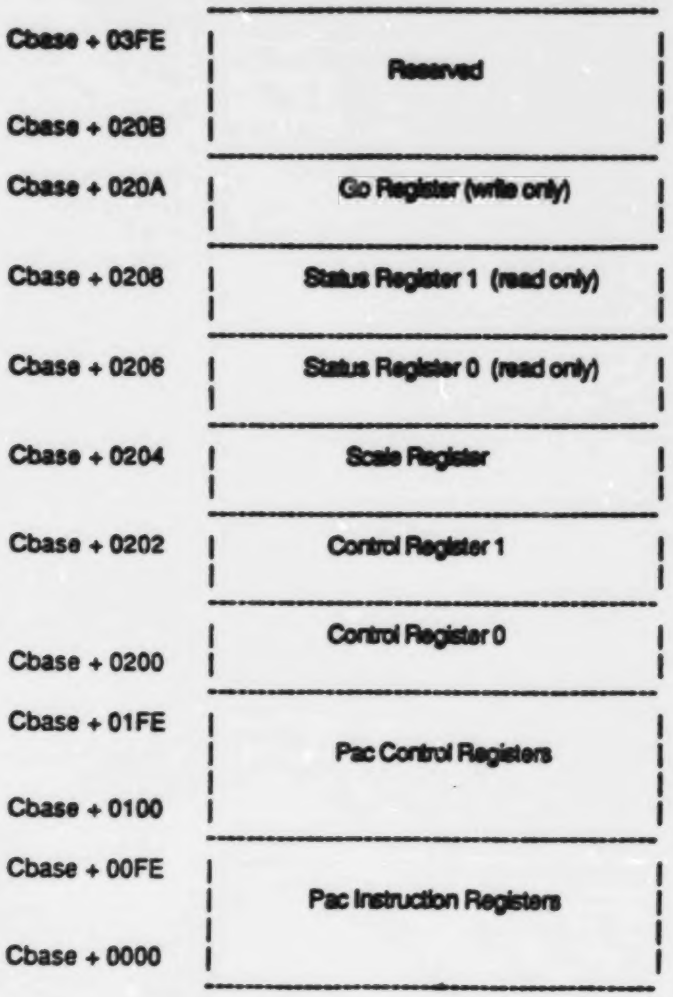

FIGURE 5.2: CONTROL AND STATUS REGISTER MEMORY MAP

5.3 PAC Registers

5.3.1 PAC Instructions

The PAC instruction memory, which may be both read to and written from, can contain 32 instructions. Each PAC instruction contains 20 bits and occupies two 16 bit words. PAC instruction bits 0 through 15 are located at addresses Cbase + 0, 8, 10, 18, etc., (Hex). PAC instruction bits 16 through 19 are located at addresses Cbase + 4, C, 14, 1C, etc.

5.3.2 PAC Control Registers

The location of the PAC control registers relative to Cbase is illustrated in Figure 5.3.

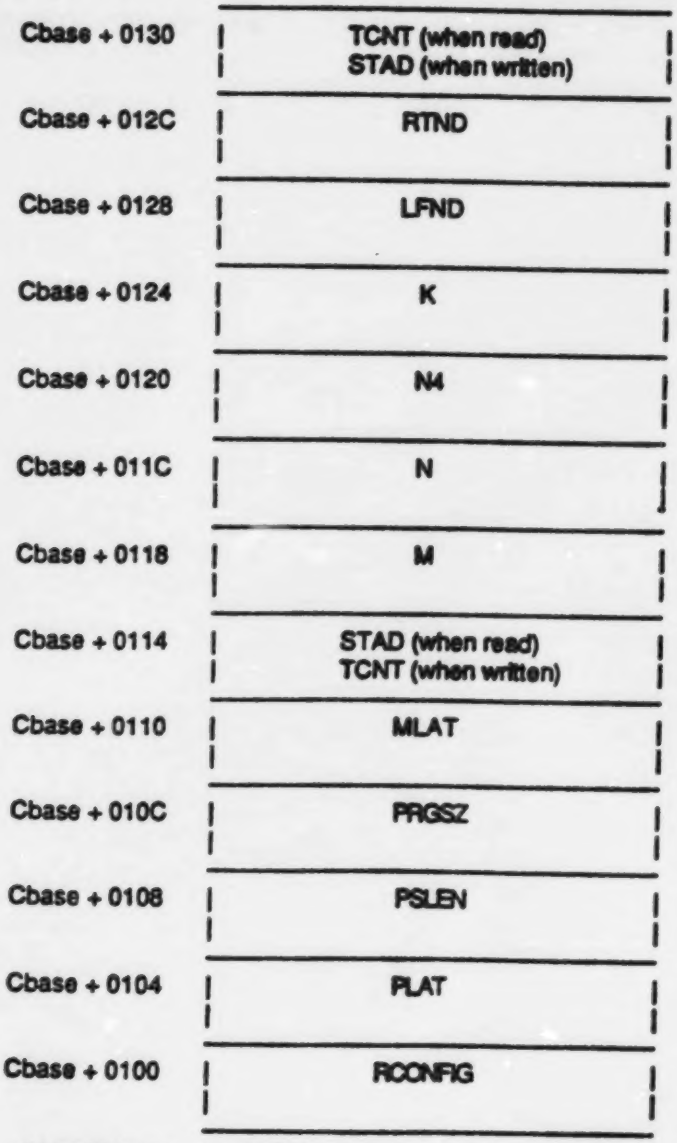

FIGURE 5.3: PAC REGISTER MEMORY MAP

These control registers allow the user to customize the PAC within the architecture in which it is configured to the user's DSP application. The control registers are defined as follows [2].

RCONFIG:
This register informs the PAC about the system configuration and the type of algorithm to be executed. This in turn defines the mode of operation for the PAC. The modes of operation include radix-2, radix-4, and mixed-mode radix FFTs, recursive, auto-start, digit-reversed write address and filter.

PLAT:
This register defines the processor latency in machine clock cycles introduced by the pipeline between the DASP input and output. PLAT must be set to four for proper operation of the DASP as configured in the FDAP architecture.

PSLEN:
This register contains the user-specified pause length, in machine clock cycles, required at the completion of a pass. This register is used when the PAC is operating in the auto-restart mode. The Auto-Restart bit is in the RCONFIG register. This bit configures the PAC to enter the pause mode at the completion of a pass before executing the next instruction.

PRGSZ:
This register defines the size of the program which has been loaded into PAC internal instruction memory. The upper limit on PRGSZ is 32.

MLAT:
This register defines the memory latency which is present in the data and address paths of the read, write, and auxiliary data memories.

STAD:
This register defines the starting address, or page offset required when addressing N points within a frame of data.

TCNT:
This register is for test purposes.

M:
This register contains the exponent of the radix used in the FFT which satisfies the definition for data array size N=(radix)**M.

N:
This register defines the size of the data array and is defined as N=(radix)**M.

M4:
This register is used to generate the appropriate address sequencing for the mixed-radix FFT mode. It determines the half array size and is defined as (M-1)/2.

K:
This register defines the overlap size required for the fast convolution overlap/discard digital filtering algorithm.

LFND and RTND:
These registers are used for defining the left and right neighbouring nodes if multiple FDAPs are used in a cascaded system.

5.4 Instruction Format

Each PAC instruction consists of 20 bits partitioned into five fields as shown in Figure 5.4 [2]. The fields are node type, mixed-radix mode, auxiliary address shift factor, bus switch code, and programmable output. The node type field is discussed in Section 5.4.1. The mixed-radix mode bit is used when a mixed-mode (combination radix-2, radix-4) FFT is to be implemented. The array size for a mixed-mode FFT is an odd power of 2. The mixed-radix mode bit allows the same sine/cosine memories to be used for the various passes in the mixed-mode FFT. The auxiliary shift factor defines the number of bit positions the auxiliary address is to be shifted to the left for one mode with respect to the other. The bus switch code defines how the input, output, read and write address generators are routed to address ports ADRA, ADRB, ADRC and ADRD. The programmable output is discussed in Section 5.4.2.

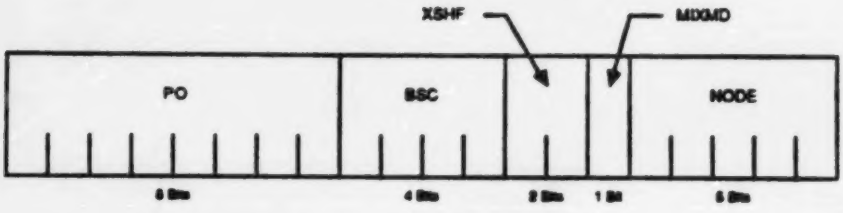

FIGURE 5.4: PAC INSTRUCTION FORMAT

5.4.1 Node Type Field

The 5-bit node type field in the PAC instruction specifies the type of address sequence, sequential or FFT specific, required of the read, write and auxiliary address generators. Table 5.1 [2] lists all of the node types supported, their mnemonics and their corresponding Hex codes for the 5 bit field.

HEX CODE	MNEMONICS	DESCRIPTION
00	FFT0	FFT Column 0
01	FFT1	FFT Column 1
02	FFT2	FFT Column 2
03	FFT3	FFT Column 3
04	FFT4	FFT Column 4
05	FFT5	FFT Column 5
06	FFT6	FFT Column 6
07	FFT7	FFT Column 7
08	FFT8	FFT Column 8
09	FFT9	FFT Column 9
0A	FFT10	FFT Column 10
0B	FFT11	FFT Column 11
0C	FFT12	FFT Column 12
0D	FFT13	FFT Column 13
0E	FFT14	FFT Column 14
0F	FFT15	FFT Column 15
10	SEQ	Sequential
11	SSEQ	Symmetric Sequential
12	FFT2N	Double Length Real FFT
13	FFTNN	Dual Real FFT
14	FTRS	Filter Sequence
15-E	RESERVED	
1F	EOPM	End of Process Marker

TABLE 5.1: PAC NODE TYPE

Function codes 00 through 0F correspond to the different columns, or passes in the DIF FFT [4] data flow diagram. The number of passes required of an FFT program depends on the radix and the size of the data array. Figures 5.5(a) and (b) [2] show the data flow diagrams of radix-4 and radix-2 FFTs, respectively. Function FFT0 executes the left most column, or first pass of the FFT. FFT(M-1) executes the right most column, or last pass of the FFT. M is the exponent that defines the array size.

Each address generator produces four distinct addresses every machine cycle. When a radix-4 FFT is implemented these addresses are used for the 4-point butterfly computation. When a radix-2 FFT is implemented, the first 2 addresses are used for the first 2-point butterfly computation, and the next two are used for the second 2-point butterfly computation each time the loop is processed.

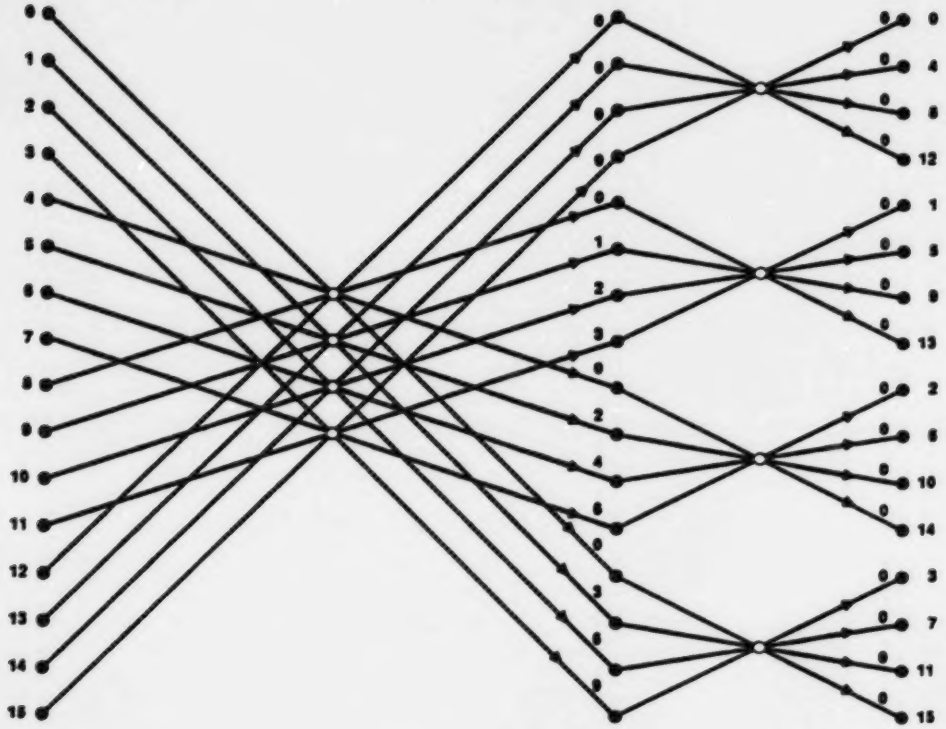

FIGURE 5.5(a): RADIX-4, 16 POINT DIF FFT FLOW DIAGRAM

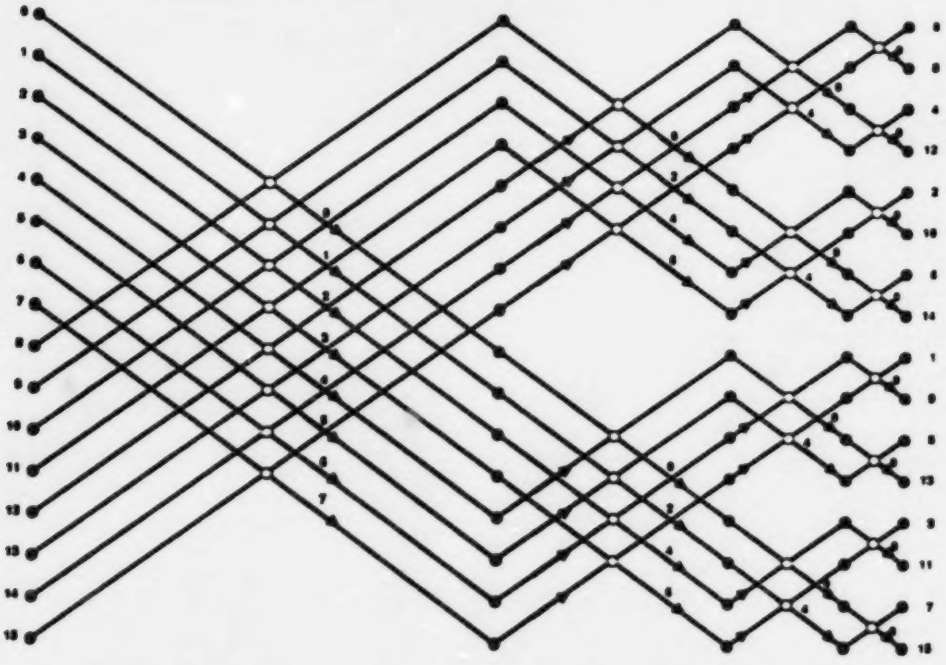

FIGURE 5.5(b): RADIX-2, 16 POINT DIF FFT FLOW DIAGRAM

5.4.2 Function Set

The programmable output field is used to specify the function code for the DASP. The remaining bits may be used to manage other parts of the system. For example, one bit may be used to select between the two auxiliary memories for the auxiliary data source. The operational 4-bit code, a mnemonic and a description of each function which is supported on the DASP are given in Table 5.2 [1].

Function Mnemonic	Opcode	Description
Complex Arithmetic Class		
BLFY4	0000	RADIX-4 DIF Butterfly
BFLY2	0001	Two RADIX-2 DIF Butterflies
FFT2N	0010	Recombine N Complex-Point to 2N Real-Point FFT
FFTNN	0011	Recombine N Complex-Point FFT to two N Real Point FFTs
BMUL	0101	Block Multiply two Sets of Complex number
General Arithmetic Class		
AFLOW	0100	Arithmetic Flow Through
BSQSM	0110	Block Square and Sum a Set of Complex Values
BADD	0111	Block Add Two Sets of Real or Complex Values
BSUB	1000	Block Subtract Two Sets of Real or Complex Values
BMULR	1001	Block Multiply Two Sets of Real Numbers
BMULRA	1011	Block Multiply Two Sets of Real Numbers and Partially Add
General Logic Class		
BCONS	1010	Generate a Block of Constants (0 or 1)
LFLOW	1100	Logical Flow-Through (Pass Data)
BAND	1101	Block AND Two Sets of Integer Values
BOR	1110	Block OR Two Sets of Integer Values
BXOR	1111	Block EXCLUSIVE-OR Two Sets of Integer Values

TABLE 5.2: DASP FUNCTION SET

6.0 FDAP PROGRAMMING USING A USER INTERFACE

The following sections outline the implementations of two DSP routines on the I/O buffered recursive system of the FDAP. PAC programs implementing FFTs and pulse compression routines follow. The examples can be implemented through either the IBM PC-AT simulator or the user interface for the EVM.

The DASP function codes are as defined in the previous sections. These programs are coded to allow the PAC to loop through the algorithm continuously until it reaches the end of the program.

6.1 Radix-4 FFT Program

In this example, memories A and/or B can be loaded with 1024 complex words of data generated by the simulator program. We have chosen to load memory B. The PAC control registers and PAC program memory are set up to do a 5 pass (1024 point), radix-4, complex point FFT on each data set. The program control registers are initialized to handle the scale factors appropriately. The PAC program is executed, and the results in memories D and/or C, C in our case, are plotted.

The simulator generates data according to the simple time domain function formulated below [5]:

$$X_i = 2^Q \left[10^{\frac{A1}{20}} e^{\frac{j2\pi F1i}{N}} + 10^{\frac{A2}{20}} e^{\frac{j2\pi F2i}{N}} \right] \qquad (6.1)$$

where:

- N = the number of samples,
- i = 0..N-1,
- A1 = the amplitude of the first tone (dB),
- F1 = the frequency bin of the first tone,
- A2 = the amplitude of the second tone (dB),
- F2 = the frequency bin of the second tone, and
- Q = the number of bits (of a tone of amplitude of 0 dB) .

The following parameters are used in our example for memory B:

N = 1024
A1 = 0 dB
F1 = 100
A2 = -6 dB
F2 = 800
Q = 12 .

A Harris-Blackman window is loaded into Auxiliary Memory X2 via the user interface. Likewise, the twiddle factors for the FFT are loaded into Auxiliary Memory X1. The PAC control registers need to be initialized as follows to perform a 1024 complex point, windowed radix-4 FFT:

RCONFIG = 5800H
PLAT = 4H
PSLEN = 0
PRGSZ = 10H
MLAT = 000AH
STAD = 0
M = 5
N = 400H
M4 = 0 (don't care)
K = 0 (don't care) .

The following instruction codes are loaded into the PAC internal instruction memory.

INSTRUCTION #	INSTRUCTION	DESCRIPTION
0	45010	BMUL; Coefficient Mem.; BSC=0; SEQ
1	00100	BFLY4; Twiddle Mem.; BSC=1; FFT0
2	00001	BFLY4; Twiddle Mem.; BSC=0; FFT1
3	00102	BFLY4; Twiddle Mem.; BSC=1; FFT2
4	00003	BFLY4; Twiddle Mem.; BSC=0; FFT3
5	00104	BFLY4; Twiddle Mem.; BSC=1; FFT4
6	04010	AFLOW; BSC=0; SEQ
7	0401F	BSC=0; EOPM
8	45310	BMUL; Coefficient Mem.; BSC=3; SEQ
9	00400	BFLY4; Twiddle Mem.; BSC=4; FFT0
A	00301	BFLY4; Twiddle Mem.; BSC=3; FFT1
B	00402	BFLY4; Twiddle Mem.; BSC=4; FFT2
C	00403	BFLY4; Twiddle Mem.; BSC=3; FFT3
D	00404	BFLY4; Twiddle Mem.; BSC=4; FFT4
E	04310	AFLOW; BSC=4; SEQ
F	0431F	BSC=4; EOPM

A window operation is first performed on the data array contained in memory B by way of instruction 0 (programmable output bit 6 is set to select the coefficient memory). Instructions 1 through 5 subsequently execute the 5 passes of the DIF FFT. Upon the completion of instruction 5 the processed data array is stored in memory B. This is specified by the BSC field of the instruction. The FDAP system is designed to output data through either memory C or memory D. Instruction 6 thus executes a flow through pass to transfer the data array in memory B to memory C. Instruction 7 then executes an End of Process Marker. This instruction executes what is referred to as a pseudo-pass and synchronizes the input and output operations on memories A and D with the end of the FFT algorithm. The EOPM instruction causes the processor to initialize an interrupt signal for one machine cycle. The DASP then performs its housecleaning before a new process is started. Instructions 8 through F execute another FFT. This time data memories A and D are made use of for the FFT.

The time domain data from the simulator which is loaded into Memory B is illustrated in Figure 6.1. The twiddle factors for the FFT, which are loaded into Auxiliary Memory X1, are given in Figure 6.2. The Harris-Blackman window, which is loaded into Auxiliary Memory X2, is shown in Figure 6.3. Finally, the FFT of the input data, which is obtained from Memory C, is given in Figure 6.4.

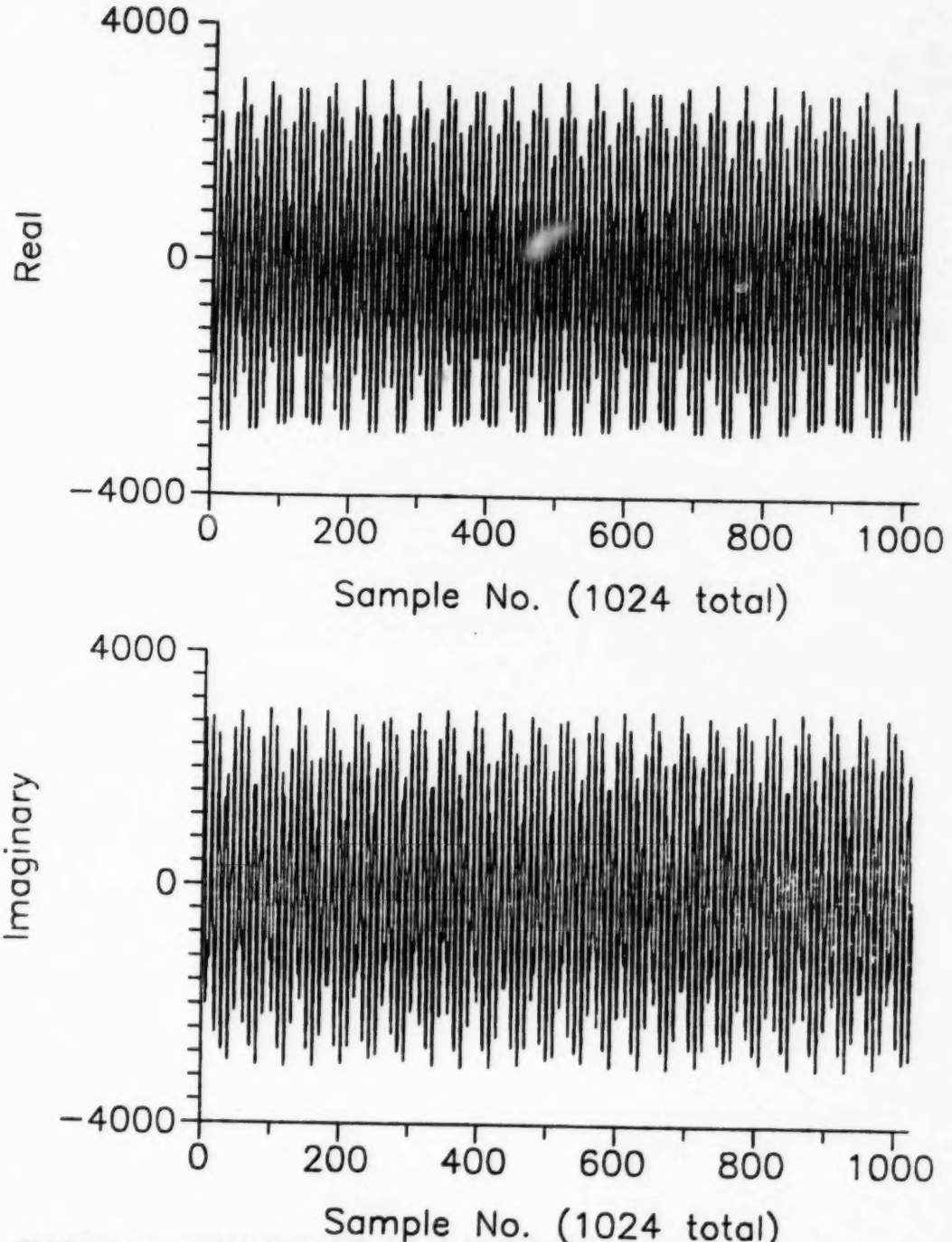

FIGURE 6.1: TIME DOMAIN DATA FROM SIMULATOR (MEMORY B)

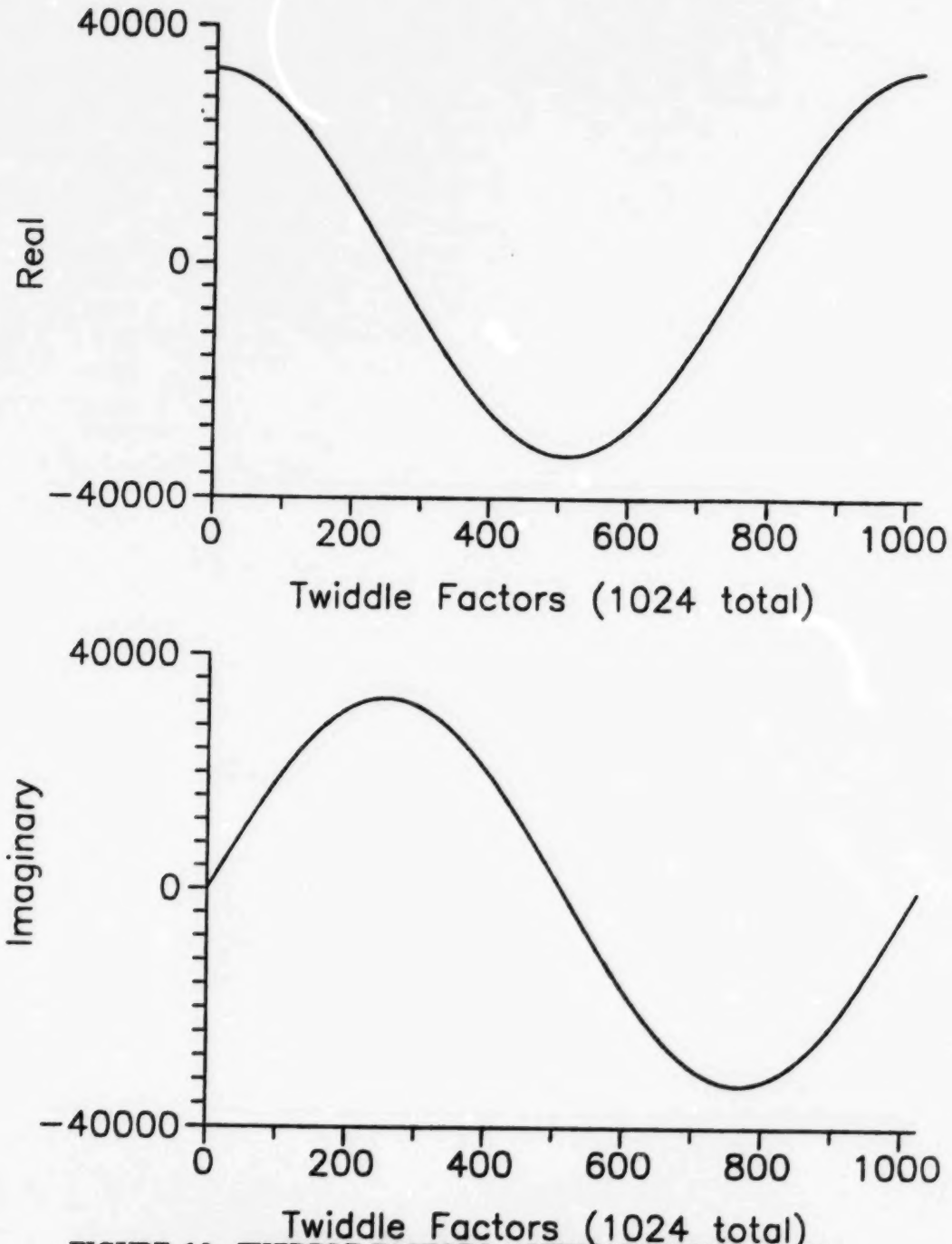

FIGURE 6.2: TWIDDLE FACTORS (AUXILIARY MEMORY X1)

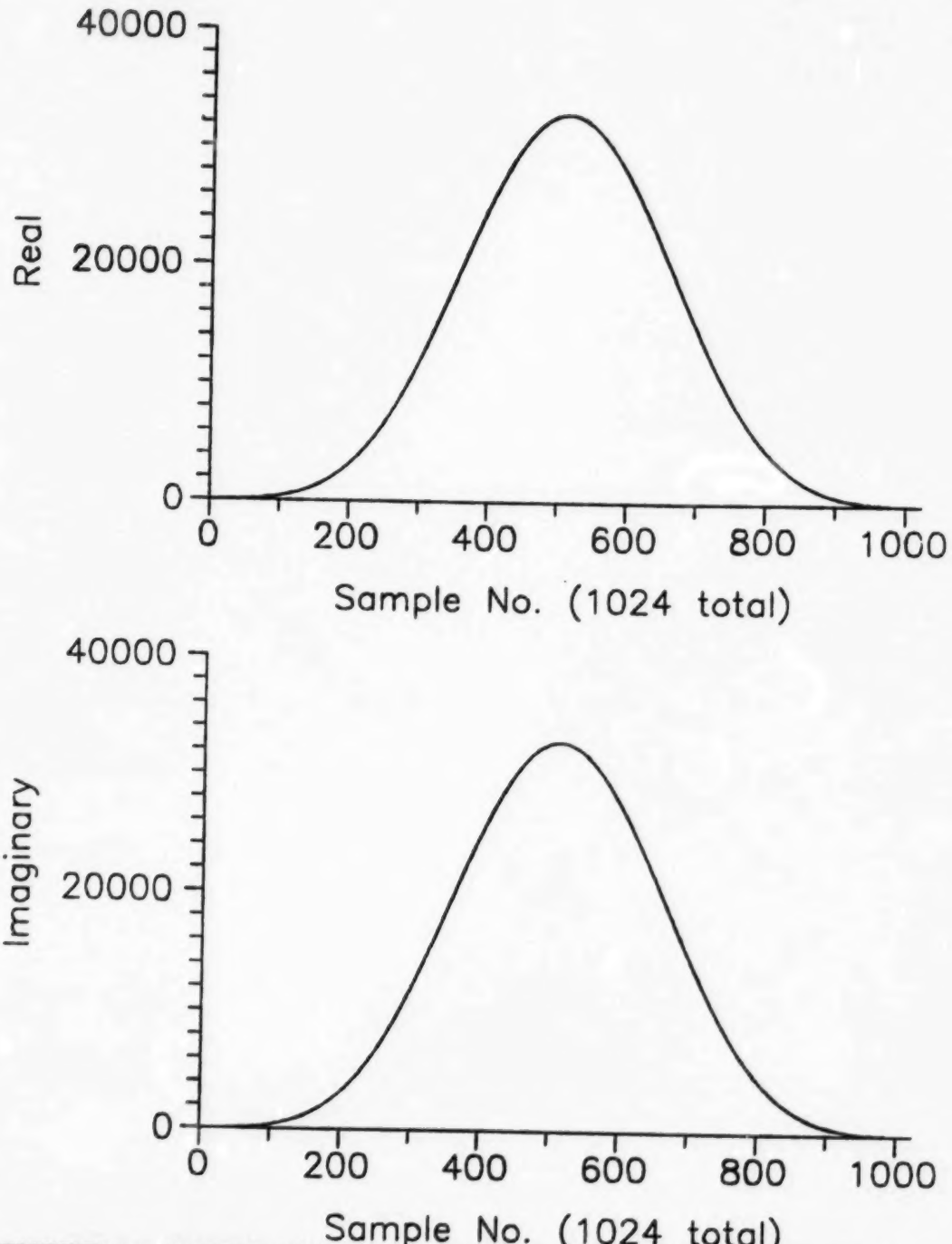

FIGURE 6.3: HARRIS-BLACKMAN WINDOW (AUXILIARY MEMORY X2)

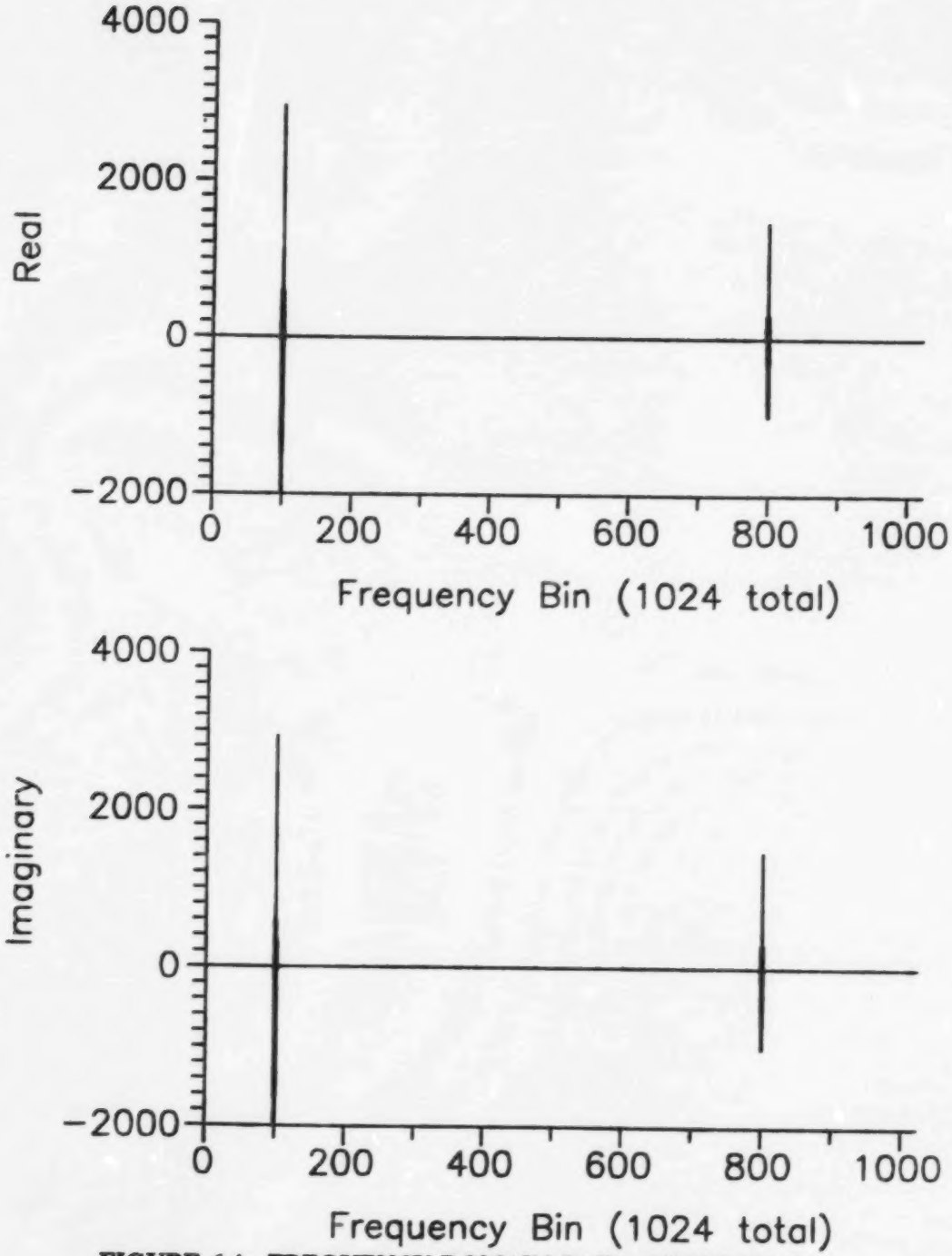

FIGURE 6.4: FREQUENCY DOMAIN DATA (MEMORY C)

6.2 Pulse Compression Program Using Radix-4 FFTs

6.2.1 Theory of Pulse Compression

A radar system requiring both long detection range and fine range resolution must transmit extremely narrow pulses of exceptionally high peak power. The practical limits on the level of peak power that can be used, however, limit how narrow a pulse can be. To obtain long detection ranges at PRFs low enough for pulse delay ranging, fairly wide pulses must be transmitted. One solution to this dilemma is pulse compression [3]. Pulse compression entails the transmission of an internally modulated pulse of sufficient width to provide the necessary average power at a reasonable level of peak power followed by the compression of the received echoes through the decoding of their modulation.

One of the most common modulated pulses used is the linear Frequency Modulated (FM) or chirp pulse. The radio frequency of a transmitted chirp pulse increases at a constant rate throughout its duration. Every echo, naturally, has the same linear increase in frequency. The received echoes are passed through a filter which introduces a time lag that decreases linearly with frequency at exactly the same rate as the frequency of the echoes increases. Being of progressively higher frequency, the trailing portion of an echo takes less time to pass through than the leading portion. Successive portions thus tend to bunch up. Consequently, when the pulse emerges from the filter its amplitude is much greater and its width much less than when it entered. The pulse has been compressed. In a digital computer pulse compression is achieved by fast convolution [4]. The Fourier Transform of the received pulse is multiplied by the frequency response of the matched filter and the result is then Inverse Fourier Transformed. This process is illustrated in Figure 6.5.

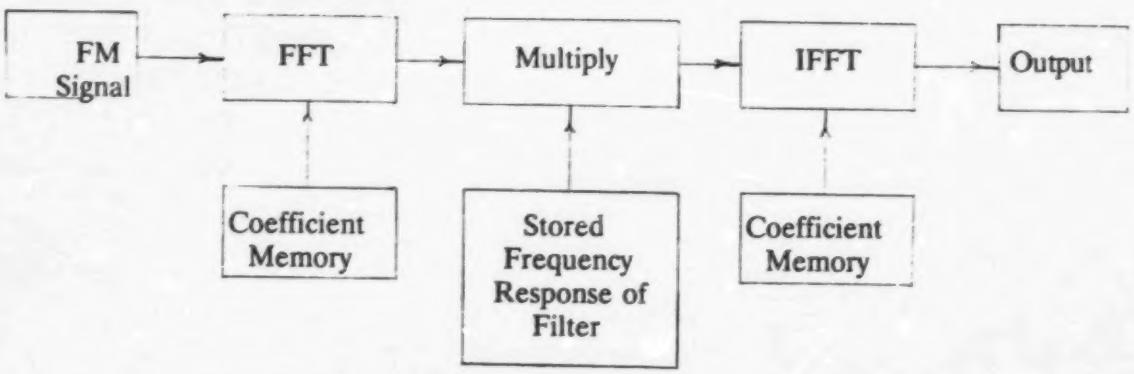

FIGURE 6.5 PULSE COMPRESSION ALGORITHM

6.2.2 Pulse Compression On The FDAP Board

All methods of pulse compression are essentially matched filtering schemes in which the transmitted pulse is coded and the received pulse is passed through a filter whose impulse response is proportional to the conjugate of the time reversed signal.

Since the FDAP software is limited in terms of signal generation, a linear frequency modulated signal has been generated externally and imported into the FDAP software. The formulation for the linear FM signal and the matched filter impulse response are as follows:

Linear FM Signal

$$s(t) = e^{j2\pi (f_o t + kt^2/2)} \tag{6.2}$$

Matched Filter Impulse Response

$$s^*(-t) = e^{j2\pi (f_o(-t) + k(-t)^2/2)} = e^{-j2\pi (f_o t - kt^2/2)} \tag{6.3}$$

where f_o is the initial frequency of the pulse and k is rate of change of the carrier frequency. f_o = 0 Hz, k = 400 Hz/second, and t = 0, 1/N, 2/N, ..., (N-1)/N where N=1024 for our example.

6.2.3 Pulse Compression Program

The PAC control registers need to be initialized as follows to compress a 1024-complex point linear FM pulse. The FFTs used are implemented with a radix-4 algorithm.

RCONFIG	= 5800H
PRGSZ	= BH
STAD	= 0
M	= 5
N	= 400H
M4	= 0
K	= 0

The following program is loaded into the PAC internal instruction memory.

INSTRUCTION #	INSTRUCTION	DESCRIPTION
0	00300	BFLY4; Twiddle Mem.; BSC=3; FFT0
1	00401	BFLY4; Twiddle Mem.; BSC=4; FFT1
2	00302	BFLY4; Twiddle Mem.; BSC=3; FFT2
3	00403	BFLY4; Twiddle Mem.; BSC=4; FFT3
4	00304	BFLY4; Twiddle Mem.; BSC=3; FFT4
5	45410	BMUL; Coefficient Mem.; BSC=4; SEQ
6	10300	Complement; BFLY4; BSC=3; FFT0
7	00401	BFLY4; Twiddle Mem.; BSC=4; FFT1
8	00302	BFLY4; Twiddle Mem.; BSC=3; FFT2
9	00403	BFLY4; Twiddle Mem.; BSC=4; FFT3
A	00304	BFLY4; Twiddle Mem.; BSC=3; FFT4

Instructions 0 through 4 execute the 5 columns of the DIF FFT on the FM signal. Instruction 5 multiplies the last FFT with the coefficient memory X2 which contains the FFT of the matched filter. Instruction 6 complements the result of the multiplication so that the inverse Fourier transform may be found using a forward FFT, and at the same time calculates FFT0. Instruction A performs the last FFT column and stores the result, the compressed pulse, in data memory D.

The following plots show the linear FM signal and the subsequent compressed pulse. The linear FM pulse, which is loaded into Memory A, is illustrated in Figure 6.6. The original pulse consists of 512 points and is zero-padded to 1024 points as required for the fast convolution. The frequency response of the matched filter, which is loaded into Auxiliary Memory X2, is given in Figure 6.7. This frequency response is computed from a 512 point impulse response which also has been zero-padded to 1024 points. Finally, the power spectrum of the compressed pulse, which is derived from the complex linear data output from Memory D, is shown in Figure 6.8. The twiddle factors are identical to those in Figure 6.2 and are stored in Auxiliary Memory X1.

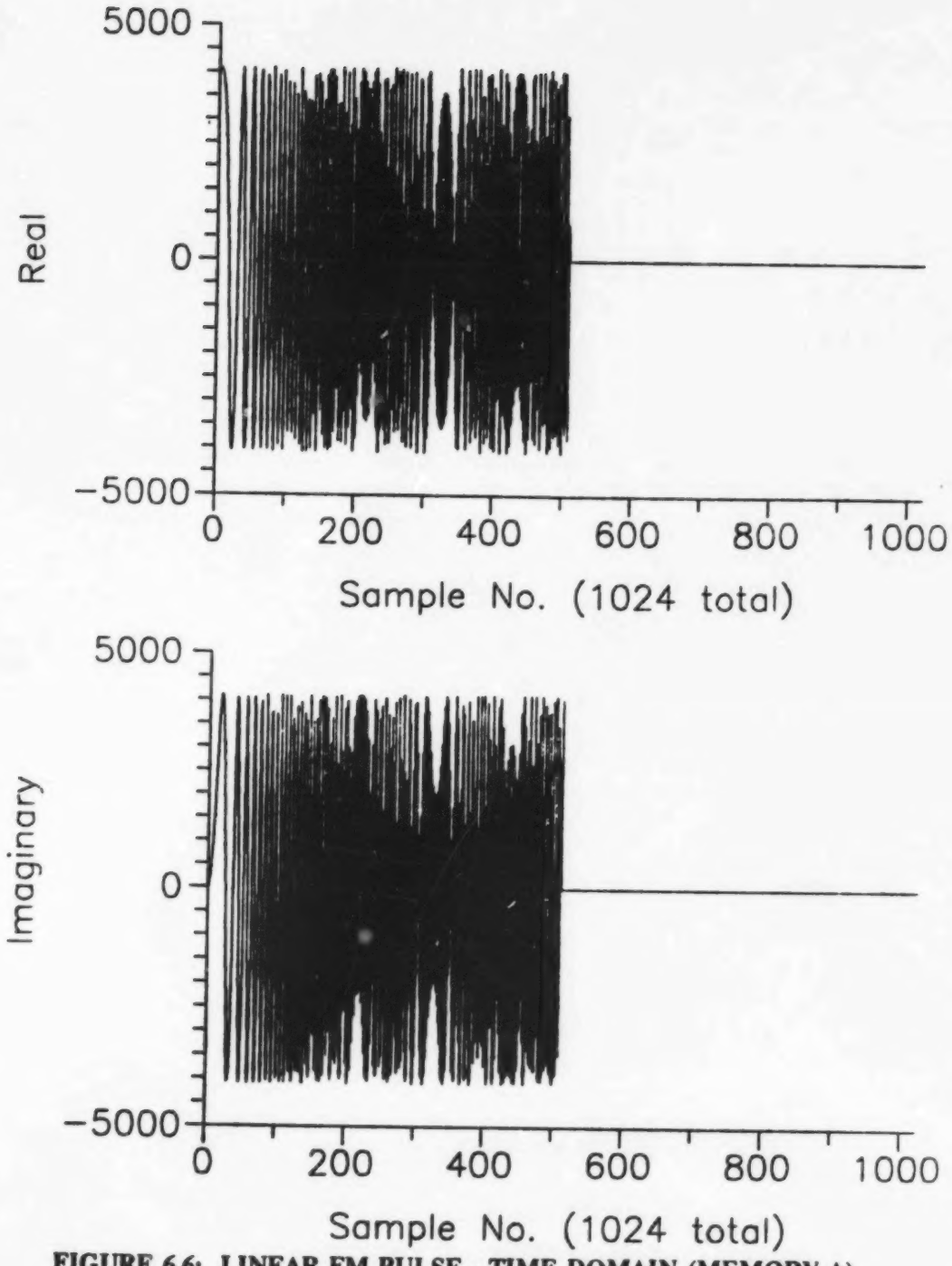

FIGURE 6.6: LINEAR FM PULSE - TIME DOMAIN (MEMORY A)

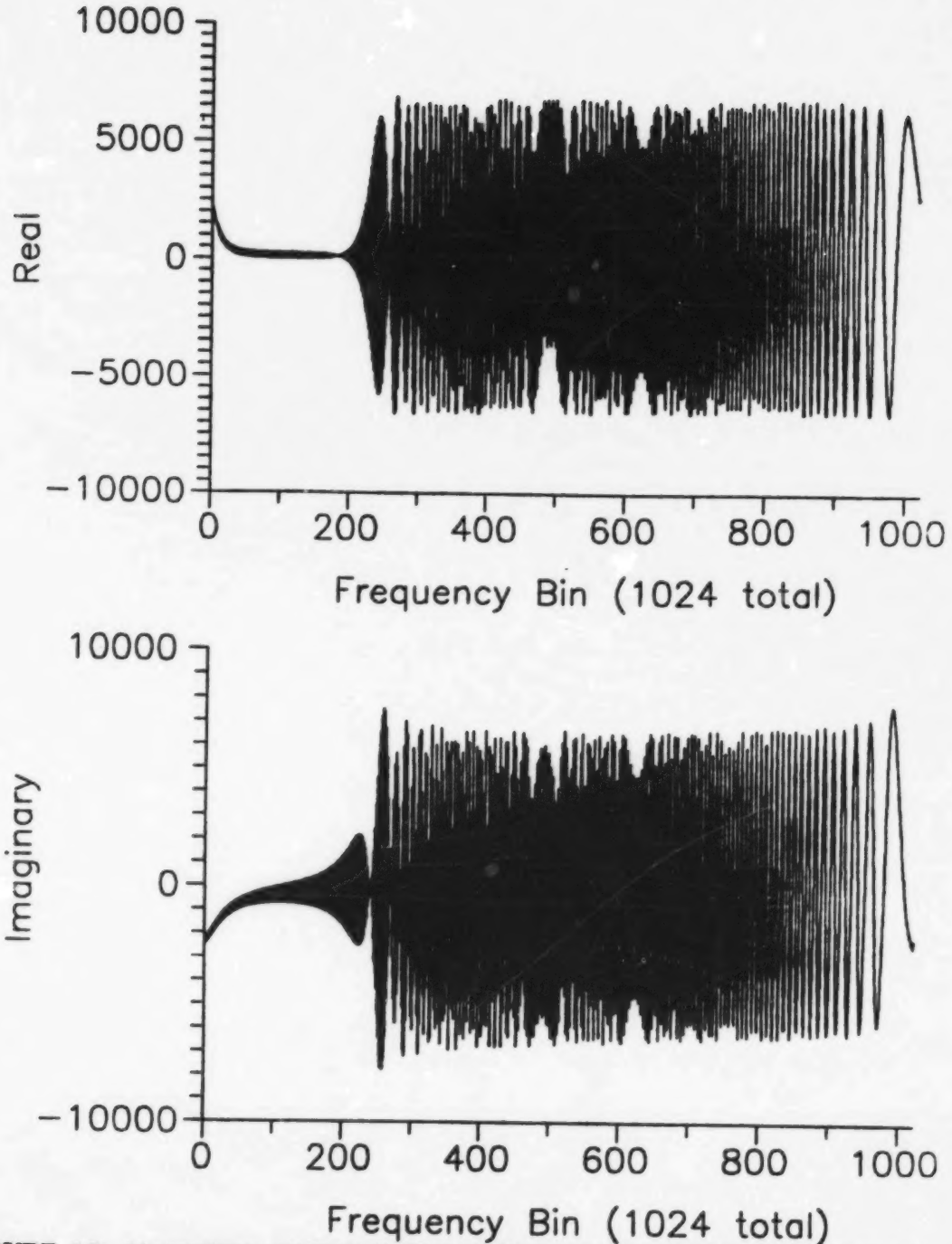

FIGURE 6.7: MATCHED FILTER IMPULSE RESPONSE - FREQUENCY DOMAIN (AUXILIARY MEMORY X2)

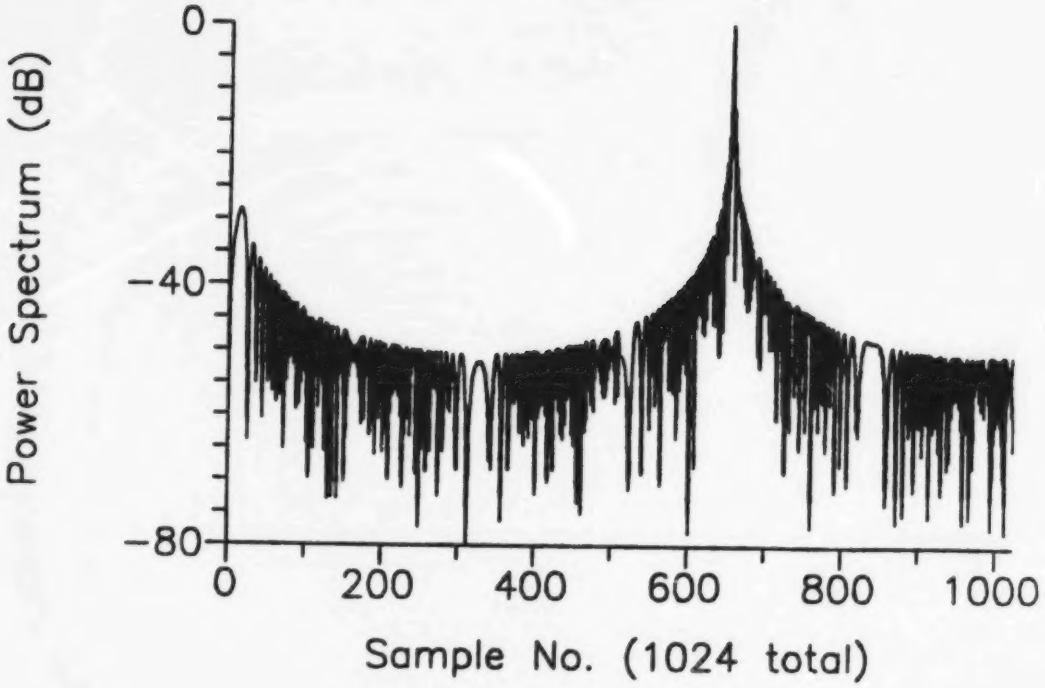

FIGURE 6.8: POWER SPECTRUM OF COMPRESSED PULSE (LINEAR DATA FROM MEMORY D)

7.0 FDAP PROGRAMMING USING A HIGH LEVEL LANGUAGE

The FDAP board has the capability to allow a user to program it externally. To allow direct access/control of the FDAP board, the control and status registers as well as memory banks have to be initialized. For a detailed explanation of the control and status bits, refer to the FDAP manual [1]. Appendix 1 contains two programs written in a high level language (C) to address the FDAP board. The first program implements a 1024-point radix-4 complex FFT on the FDAP board. The second program implements a 1024-point pulse compression routine using radix-4 FFTs. The data used in both of these programs was the same data used in Section 6.0 where the FDAP was programmed using the user-interface. Results identical to those presented in Section 6.0 were obtained.

8.0 CONCLUSIONS

The HDSP66 product family from SPT is a new generation of digital signal processing products optimized for frequency domain array processing applications. The family offers very high performance, system-based solutions for FFT applications.

The FDAP simulation package and user interface software were used to evaluate the DASP/PAC chip set. The C language was then used to implement FFT and pulse compression routines to further examine the ease of implementation of typical radar signal processing routines which could be integrated into a more sophisticated system.

It was found that the DASP/PAC chip set is a powerful signal processing tool but is not yet a totally mature product. There are no readily available signal processing routines for the FDAP (such as a subroutine library) so that the board must be accessed using the low-level calls presented in the documentation. The documentation itself is difficult to use and thus the development time is extensive. The C-language programs implemented carried out their functions to the satisfaction of the users. Follow-up work would entail developing a PC-based SAR processor using the FDAP board as the main computational engine.

9.0 APPENDIX 1 - FDAP C-LANGUAGE PROGRAM

9.1 1024-Point Radix-4 Complex FFT Program

```
#include <stdio.h> /*The following program performs 1024 complex-point FFT*/
main()
{

unsigned int a, b;
int i, values, size, ch;
int huge  *mem_pointer, *aux_pointer, *con_reg0,  *con_reg1;
int huge *scl_reg, *pac_instl, *pac_instm, *pac_instm1;
int huge *pac_instm2;
int huge *rec_reg, *mlt_reg, *plt_reg, *progsz, *m, *n, *m4;
int huge *k, *stad, *fft_node_0, *  pac_instm3, *pac_instm4;
int huge *sts_reg0, *sts_reg1, *go_reg;
int huge *pslen, *tcnt, *lfnd, *rtnd;

char *infilename="c:twd.dat";/*Step to load the twiddle factor data file to auxiliary
                                memory*/
char *intype="r";
FILE *fp1;
fp1=fopen(infilename,intype);

con_reg0=0xd0200000;   /*Pointer to control register*/
*con_reg0=0x53dd;      /*Pointer to select auxiliary memory X1*/
aux_pointer=0xd8000000;

fscanf (fp1, "%X", &values);
fscanf (fp1, "%X", &a);

for (i=1; (i<=1024)&&(!feof(fp1));i+=1)
        {
        fscanf (fp1, "%X %x", &a, &b);
        *aux_pointer=a;
        aux_pointer++;
        *aux_pointer=b;
        aux_pointer++;
        }
fclose(fp1);
{               /*Steps to download data file to data memory*/
  char *infilename="c:fmhex.dat";
  char *outfilename= "c:fftfm.dat";
```

```
    char *intype="r";
    char *outtype="w";
    FILE *fp1;
    FILE *fp2;
    fp1=fopen(infilename,intype);
    fp2=fopen(outfilename,outtype);
    con_reg0= 0xd0200000;   /*Pointer to control register*/
    *con_reg0=0x53d1;       /*Pointer to select data memory bank*/
    mem_pointer=0xd8000000;
    fscanf (fp1, "%X", &values);
    fscanf (fp1, "%X", &a);

for (i=1; (i<=1024)&&(!feof(fp1));i+=1)
        {
        fscanf (fp1, "%X  %X", &a, &b);

        *mem_pointer=a;
        mem_pointer++;
        *mem_pointer=b;
        mem_pointer++;
        }
fclose(fp1);
mem_pointer=0xd8000000;
        for (i=1; (i<=1024);i+=1)
        {
        mem_pointer+=2;
        fprintf (fp2, "%d %d\n", *(mem_pointer-2), *(mem_pointer-1));
        }
        fclose(fp2);

                        /*Pointer to different control register*/
         con_reg0  = 0xd0200000;
         con_reg1  = 0xd0200002;
         scl_reg   = 0xd0200004;
         pac_instl = 0xd0000000;
         pac_instm = 0xd0000004;
         pac_instm1= 0xd000000c;
         pac_instm2= 0xd0000014;
         pac_instm3= 0xd000001c;
         pac_instm4= 0xd0000024;
         rec_reg   = 0xd0100000;
         plt_reg   = 0xd0100004;
         mlt_reg   = 0xd0110000;
         progsz    = 0xd010000c;
```

```
m    = 0xd0110008;
n    = 0xd011000c;
m4        = 0xd0120000;
k         = 0xd0120004;
stad      = 0xd0130000;
sts_reg0  = 0xd0200006;
sts_reg1  = 0xd0200008;
go_reg    = 0xd020000a;
pslen     = 0xd0100008;
tcnt      = 0xd0110004;
lfnd      = 0xd0100028;
rtnd      = 0xd012000c;

                    /*steps to initialize control registers*/
*con_reg0  = 0x23d0;
*con_reg1  = 0x140;
*scl_reg   = 0xff00;
*pac_instl = 0x0;
*pac_instm = 0x300;
*pac_instm1= 0x401;
*pac_instm2= 0x302;
*pac_instm3= 0x403;
*pac_instm4= 0x304;
*rec_reg   = 0x4800;
*plt_reg   = 0x4;
*mlt_reg   = 0x2a;
*pslen     = 0x0;
*tcnt      = 0x0;
*lfnd      = 0x0;
*rtnd      = 0x0;
*progsz    = 0x5;
*m         = 0x5;
*n         = 0x400;
*m4        = 0x0;
*k         = 0x0;
*stad      = 0x0;
*go_reg    = 0xffff;

printf ("con_reg0=%lp contents=%X\n", con_reg0, *con_reg0);
printf ("con_reg1=%lp contents=%X\n", con_reg1, *con_reg1);
printf ("scl_reg=%lp contents=%X\n", scl_reg, *scl_reg);
printf ("pac_instm=%lp contents=%X\n", pac_instm,*pac_instm);
printf ("pac_instm1=%lp contents=%X\n", pac_instm1,*pac_instm1);
printf ("pac_instm2=%lp contents=%X\n", pac_instm2,*pac_instm2);
```

```
printf ("pac_instm3 = %lp contents = %X\n", pac_instm3,*pac_instm3);
printf ("pac_instm4 = %lp contents = %X\n", pac_instm4,*pac_instm4);
printf ("rec_reg = %lp contents = %X\n", rec_reg, *rec_reg);
printf ("progsz = %lp contents = %X\n", progsz, *progsz);
printf ("m = %lp contents = %X\n", m, *m);
printf ("n = %lp contents = %X\n", n, *n);
printf ("go_reg = %lp contents = %X\n", go_reg, *go_reg);
printf ("sts_reg0 = %lp contents = %X\n", sts_reg0, *sts_reg0);
printf ("sts_reg1 = %lp contents = %X\n", sts_reg1, *sts_reg1);

con_reg0    = 0xd0200000;
*con_reg0   = 0x7;
mem_pointer = 0xd8000000;
  for (i=0; i<=32768; i++)
  {
  printf ("Memory Address = %lp Contents = %X\n", mem_pointer, *mem_pointer);
  mem_pointer++;
  }
}
}
```

9.2 1024-Point Radix-4 Pulse Compression Program

```
/* This program is set to initializes the FDAP control
   registers, data memory, and instruction registers to perform pulse compression.*/

    #include <stdio.h>
    main()
    {
    unsigned int a, b;
    int i,  values, size, ch;
    int huge  *mem_pointer, *aux_pointer, *con_reg0;
    int huge *con_reg1;
    int huge  *scl_reg;
    int huge *pac_inst0, *pac_inst1, *pac_inst2, *pac_inst3;
    int huge  *pac_instm0, *pac_instm1, *pac_instm2;
    int huge *pac_instm5;
    int huge  *pac_instm6, *pac_instm7, *pac_instm8;
    int huge *pac_instm9;
    int huge  *pac_instm10;
    int huge  *rec_reg, *mlt_reg, *plt_reg, *progsz, *m, *n;
    int huge *m4;
    int huge  *k, *stad, *fft_node_0, *pac_instm3;
    int huge *pac_instm4;
    int huge  *sts_reg0, *sts_reg1, *go_reg;
    int huge  *pslen, *tcnt, *lfnd, *rtnd;

    char *infilename="c:firout.dat";/*Steps to set the auxiliary memory X2*/
    char *outfilename= "c:mfout.dat";
    char *intype="r";
    char *outtype="w";

 FILE *fp1;
 FILE *fp2;
 fp1=fopen(infilename,intype);
 fp2=fopen(outfilename,outtype);
   con_reg0= 0xd0200000;
   *con_reg0=0x53df;
   mem_pointer=0xd8000000;
fscanf (fp1, "%X", &values);
fscanf (fp1, "%X", &a);

 for (i=1; (i<=1024)&&(!feof(fp1));i+=1)
     {
     fscanf (fp1, "%X  %X", &a, &b);
```

```
        *mem_pointer=a;
        mem_pointer++;
        *mem_pointer=b;
        mem_pointer++;
        }
        fclose(fp1);
mem_pointer=0xd8000000;
for (i=1; (i<=1024);i+=1)
{
mem_pointer+=2;
fprintf (fp2, "%x %x\n", *(mem_pointer-2),*(mem_pointer-1));
}
        fclose(fp2);

{
   char *infilename="c:twd.dat";  /*Steps to set X1*/
   char *intype="r";
FILE *fp1;
fp1=fopen(infilename,intype);

con_reg0=0xd0200000;
*con_reg0=0x53dd;
aux_pointer=0xd8000000;

fscanf (fp1, "%X", &values);
fscanf (fp1, "%X", &a);

for (i=1; (i<=1024)&&(!feof(fp1));i+=1)
   {
   fscanf (fp1, "%X %x", &a, &b);
   *aux_pointer=a;
   aux_pointer++;
   *aux_pointer=b;
   aux_pointer++;
   }
fclose(fp1);
{
   char *infilename="c:fmhex.dat"; /*step to load data memory*/
   char *outfilename= "c:fftfm.dat";
   char *intype="r";
   char *outtype="w";
   FILE *fp1;
   FILE *fp2;
   fp1=fopen(infilename,intype);
```

```
    fp2=fopen(outfilename,outtype);

    con_reg0= 0xd0200000;
    *con_reg0=0x53d1;
    mem_pointer=0xd8000000;

    fscanf (fp1, "%X", &values);
    fscanf (fp1, "%X", &a);

    for (i=1; (i<=1024)&&(!feof(fp1));i+=1)
    {

    fscanf (fp1, "%X  %X", &a, &b);

    *mem_pointer=a;
    mem_pointer++;
    *mem_pointer=b;
    mem_pointer++;
}
fclose(fp1);

   mem_pointer=0xd8000000;
 for (i=1; (i<=1024);i+=1)
{
   mem_pointer+=2;

fprintf (fp2, "%d %d\n", *(mem_pointer-2), *(mem_pointer-1));
}
fclose(fp2);                    /*Pointer to control registers*/

       con_reg0 = 0xd0200000;
       con_reg1 = 0xd0200002;
       scl_reg =  0xd0200004;
       pac_inst0= 0xd0000000;
       pac_instm0=0xd0000004;
       pac_instm1=0xd000000c;
       pac_instm2=0xd0000014;
       pac_instm3=0xd000001c;
       pac_instm4=0xd0000024;
       pac_inst1= 0xd0000028;
       pac_instm5=0xd000002c;
       pac_inst2= 0xd0000032;
       pac_instm6=0xd0000034;
       pac_inst3= 0xd0000038;
```

```
        pac_instm7=0xd000003c;
        pac_instm8=0xd0000044;
        pac_instm9=0xd000004c;
        pac_instm10=0xd0000054;
        rec_reg =  0xd0100000;
        plt_reg =  0xd0100004;
        mlt_reg =  0xd0110000;
        progsz  =  0xd010000c;
        m =  0xd0110008;
        n =  0xd011000c;
        m4 =  0xd0120000;
        k =  0xd0120004;
        stad =  0xd0130000;
        sts_reg0 =  0xd0200006;
        sts_reg1 =  0xd0200008;
        go_reg  =  0xd020000a;
        pslen   =  0xd0100008;
        tcnt    =  0xd0110004;
        lfnd    =  0xd0100028;
        rtnd    =  0xd012000c;
                              /*Steps to initialize control registers*/
*con_reg0 = 0x53d0;
*con_reg1 = 0x140;
*scl_reg  = 0xff00;
*pac_inst0=0x0;
*pac_instm0 =0x300;
*pac_instm1=0x401;
*pac_instm2=0x302;
*pac_instm3=0x403;
*pac_instm4=0x304;
*pac_inst1=0x4;
*pac_instm5=0x5410;
*pac_inst2=0x1;
*pac_instm6=0x300;
*pac_inst3=0x0;
*pac_instm7=0x401;
*pac_instm8=0x302;
*pac_instm9=0x403;
*pac_instm10=0x304;
*rec_reg =0x4800;
*plt_reg =0x4;
*mlt_reg =0x2a;
*pslen=0x0;
*tcnt =0x0;
```

```
    *lfnd=0x0;
    *rtnd=0x0;
    *progsz=0xb;
    *m = 0x5;
    *n = 0x400;
    *m4= 0x0;
    *k = 0x0;
    *stad = 0x0;
    *go_reg = 0xffff;
printf ("con_reg0=%lp contents=%X\n", con_reg0, *con_reg0);
printf ("con_reg1=%lp contents=%X\n", con_reg1, *con_reg1);
printf ("scl_reg=%lp contents=%X\n", scl_reg, *scl_reg);
printf ("pac_inst0=%lp contents=%X\n", pac_inst0, *pac_inst0);
printf ("pac_instm=%lp contents=%X\n", pac_instm0, *pac_instm0);
printf ("pac_instm1=%lp contents=%X\n", pac_instm1, *pac_instm1);
printf ("pac_instm2=%lp contents=%X\n", pac_instm2, *pac_instm2);
printf ("pac_instm3=%lp contents=%X\n", pac_instm3, *pac_instm3);
printf ("pac_instm4=%lp contents=%X\n", pac_instm4, *pac_instm4);
printf ("pac_inst1=%lp contents=%X\n", pac_inst1, *pac_inst1);
printf ("pac_instm5=%lp contents=%X\n", pac_instm5, *pac_instm5);
printf ("pac_inst2=%lp contents=%X\n", pac_inst2, *pac_inst2);
printf ("pac_instm6=%lp contents=%X\n", pac_instm6, *pac_instm6);
printf ("pac_instm7=%lp contents=%X\n", pac_instm7, *pac_instm7);
printf ("pac_instm8=%lp contents=%X\n", pac_instm8, *pac_instm8);
printf ("pac_instm9=%lp contents=%X\n", pac_instm9, *pac_instm9);
printf ("pac_instm10=%lp contents=%X\n", pac_instm10, *pac_instm10);
printf ("rec_reg=%lp contents=%X\n", rec_reg, *rec_reg);
printf ("progsz=%lp contents=%X\n", progsz, *progsz);
printf ("m=%lp contents=%X\n", m, *m);
printf ("n=%lp contents=%X\n", n, *n);
printf ("go_reg=%lp contents=%X\n", go_reg, *go_reg);
printf ("sts_reg0=%lp contents=%X\n", sts_reg0, *sts_reg0);
printf ("sts_reg1=%lp contents=%X\n", sts_reg1, *sts_reg1);
    con_reg0=0xd0200000;
    *con_reg0=0x7;
    mem_pointer=0xd8000000;
     for (i=0; i<32768; i++)
     {
     printf ("Memory Address=%lp Contents=%X\n", mem_pointer, *mem_pointer);
     mem_pointer++;
     }
     }
     }
     }
```

10.0 REFERENCES

[1] SPT, "Digital Array Signal Processor" User's Guide, Revision 3.0, November 1,1989.

[2] SPT, "Programmable Array Controller" User's Guide, Revision 3.0, November 1, 1989.

[3] George W. Stimson, "Introduction to Airborne Radar", Hughes Aircraft Company, El Segundo, California 1983.

[4] Alan V. Oppenheim, "Digital Signal Processing", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

[5] SPT, "Frequency Domain Array Processor", User's Guide, Revision 01/90.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence Research Establishment Ottawa
Department of National Defence
Ottawa, Ontario K1A 0Z4

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

Implementation of FFT and Pulse Compression Routines on the SPT Frequency Domain Array Processor (U) .

4. AUTHORS (Last name, first name, middle initial)

Behroozi, V. Damini, A.

5. DATE OF PUBLICATION (month and year of publication of document)

September 1990

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

58

6b. NO. OF REFS (total cited in document)

5

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Report

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

Defence Research Establishment Ottawa
Department of National Defence Ottawa, K1A 0Z4

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

021LA

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DREO Report 1041

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availabilty (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both offical languages unless the text is bilingual).

The Frequency Domain Array Processor (FDAP) is a VME compatible circuit board built by Signal Processing Technologies (SPT). The FDAP can process integer data arrays containing up to 8192 (32 bit) complex words or 16384 (16 bit) real words. It is capable of 400 Million Operations Per Second (MOPS) with a maximum Input/Output (I/O) rate of four billion bits per second. It also has a double buffered memory architecture permitting I/O transfers to occur in parallel with data processing. The FDAP can be hosted by an IBM PC/AT-compatible computer using a bus adaptor interface available from BIT3 Computer Corp.

The FDAP board is based upon SPT's DASP/PAC chip set. This chip set and the various system architectures which can be built around it are reviewed. The FDAP board and its associated development system are also reviewed. The ease of implementation of typical radar signal processing functions on the FDAP board are then examined. Fast Fourier Transform and pulse compression routines are implemented via a supplied user interface as well as a high level language (C). The results are examined and comments on the FDAP and its associated system development tools are made.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Synthetic Aperture Radar
VME Bus
FFT
Pulse Compression